Progressive Web Apps von Google S. 30

Eigene Frameworks für iOS erstellen S. 64

IoT-Programmierung mit Cylon.js S. 124

web & mobile DEVELOPER

webundmobile.de

web & mobile DEVELOPER

Apps für Android, iOS und Windows 10

Plattformübergreifend entwickeln mit Qt Quick Controls 2 ab S. 14

Raspberry Pi für Entwickler

Dr. Markus Stäuble zeigt, wie man mit dem Raspberry Pi eine kompakte Entwicklungsumgebung mit allen erforderlichen Komponenten aufbaut S. 94

Ausgabe 8/16

Deutschland 14,95 EUR

CH: 29,90 CHF
A, B, NL, L: 16,45 EUR

4 198255 014951 08

Herausforderung UI

Die Programmierung ansprechender App-Oberflächen verlangt dem Entwickler einiges ab. Wie man diese Herausforderung mit Qt Quick Controls 2 meistern kann.

Qt gibt es seit mehr als 20 Jahren, und in diesem langen Zeitraum wurden verschiedene Wege verfolgt, mit Qt Oberflächen zu entwickeln. Da gab es beispielsweise Qt Widgets und Qt Quick. Alle diese Wege – auch die früheren Versionen von Quick Controls – erwiesen sich als nicht optimal. Dies war der Auslöser, die Quick Controls komplett neu zu entwickeln, ohne auf bestehende Implementierungen aufzusetzen oder an APIs gebunden zu sein. Dies war die Geburtsstunde der Qt Quick Controls 2, die jetzt auf Templates basieren und eine C++-Implementierung beinhalten. In QML erfolgt nur noch das Customizen. Unser Autor Ekkehard Gentz hat die neueste Version der Quick Controls in einem Artikel ab Seite 14 genauer unter die Lupe genommen.

»Der Raspberry Pi eignet sich hervorragend als kompakte Entwicklungsumgebung.«

Wer bereits an mehreren verschiedenen iOS-Projekten gearbeitet hat, wird sich sehr wahrscheinlich an eigenen Code erinnern, der universal genutzt werden konnte, aber womöglich zur Wiederverwendung entweder immer wieder neu geschrieben oder per Copy and Paste in neue Projekte eingebunden wurde. Beide Verfahren sind dabei weder sonderlich schön noch komfortabel. Besser ist es, derartigen Code direkt von Beginn an in ein Framework zu packen und dann zukünftig für die gewünschte Funktionalität nur noch und ausschließlich eben jenes Framework zu pflegen. Thomas Sillmann beschreibt diese Methode in einem Artikel ab Seite 64.

Der Raspberry Pi eignet sich nicht nur als ausreichend dimensionierter Desktop-Ersatz oder als Schaltzentrale fürs Smart Home. Entwickler, die sich mit diesem Minicomputer eine Entwicklungsumgebung aufbauen wollen, benötigen dafür keinen Monitor und keine Tastatur am Pi, sondern nur eine funktionierende Remote-Umgebung. Wie man diese aufbaut und wie man den Grundaufbau mit ein paar interessanten Gimmicks verfeinern kann, beschreibt Dr. Markus Stäuble in einem Artikel ab Seite 94.

Ihr Max Bold
chefredakteur@maxbold.de

Jochen Schmidt

nimmt PM2, den Prozess- und Deploymentmanager für Node.js, genauer unter die Lupe (S. 22)

Dr. Markus Stäuble

zeigt, wie man mit isolierten Docker-Containern individuelle Entwicklungsumgebungen aufbaut (S. 80)

Daniel Basler

demonstriert, wie man mit ASP.NET, Knockout.js und SignalR eine Single Page Application für das Web erstellt (S. 108)

INHALT

Aktuell

Feature

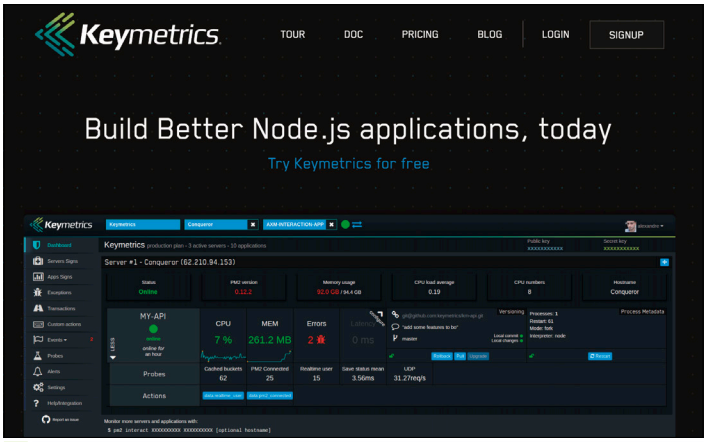
HTML, CSS & JavaScript

Mobile Development

Backend

Foto: Shutterstock / robuart

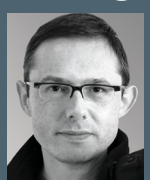
Die neuen Qt Quick Controls 2 ermöglichen es, mobile Apps für Android, iOS und Windows 10 zu bauen, die gut aussehen **S. 14**



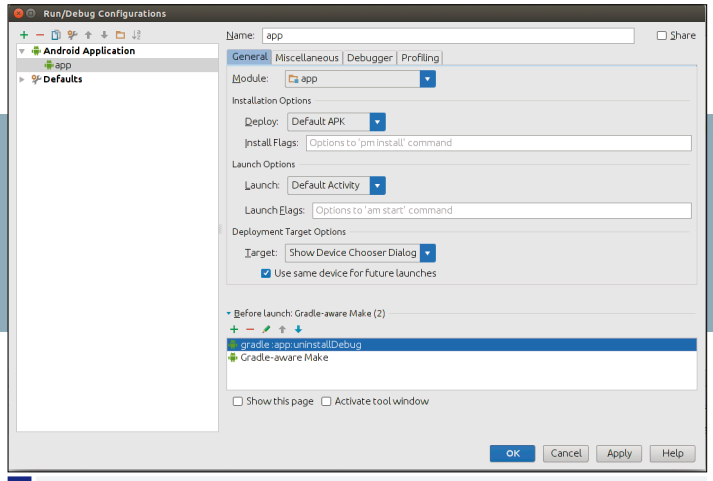
PM2 ist ein Prozess- und Deployment-Manager für Node.js mit vielen Features und einem ausgeklügelten Modulsystem **S. 22**

Experten in dieser Ausgabe

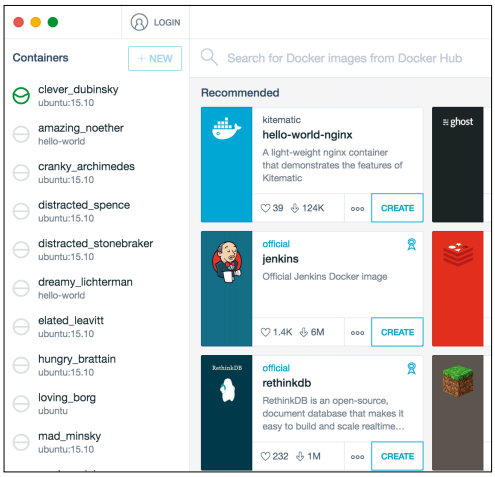
Dr. Florence Maurice stellt einen neuen Ansatz für Web-Apps vor: Progressive Web Apps von Google. **30**



Thomas Hafen erläutert, mit welcher Software man das Beste aus beiden Cloud-Welten herausholt. **88**



React-Native-Bibliotheken stellen die React-Architektur für native iOS- und Android-Applikationen zur Verfügung **S. 48**



Isolierte Container mit Docker erlauben individuelle Entwicklungsumgebungen und erleichtern damit das Deployment **S. 80**

Comics gibt es nicht erst seit gestern: Und heute sind sie bereits in allen Medien angelangt, gewissermaßen von der Antike in den Computer **S. 112**

Beyond Dev

Standards

NEWS & TRENDS

AKTUELLE NEWS FÜR ENTWICKLER

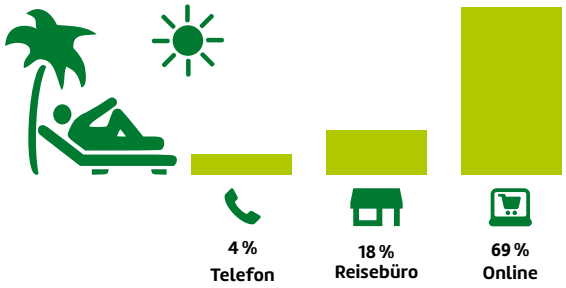
Reisebuchung

Deutsche buchen überwiegend online

Online-Reisebuchungen liegen voll im Trend: Fast sieben von zehn Deutschen (69 Prozent) wollen ihren Urlaub dieses Jahr im Internet buchen, also 4 Prozent mehr als noch 2015 (65 Prozent). Meist tun Befragte dies über den eigenen Desktop oder Laptop (62 Prozent), via Tablet-Computer (4 Prozent) oder das Smartphone (3 Prozent). Das ergab eine aktuelle Umfrage von Lastminute.de, dem Experten für Online-Reise und Freizeit.

Weit mehr als die Hälfte der Deutschen (64 Prozent) nutzt das Internet jedoch nicht nur für die Reisebuchung, sondern auch, um sich in Sachen Urlaubsdestination inspirieren zu las-

Online-Reisebuchungen liegen im Trend



Reiseplanung: So buchen Deutsche ihren Urlaub 2016.

web & mobile developer 8/2016 Quelle: Lastminute.de

sen. Im Europavergleich ist hier allerdings Italien Spitzenreiter (78 Prozent). Knapp dahinter die Spanier, von denen gut drei Viertel (76 Prozent) hier zugeben, dass das Internet sie während der Urlaubsplanung am meisten beeinflusst. In Deutschland vertraut übrigens nur jeder Zehnte (12 Prozent) in Sachen Urlaubstipps und Reiseempfehlungen auf Freunde, Kollegen und Familie. Ebenfalls weit mehr als die Hälfte (60 Prozent) aller Befragten in Großbritannien glaubt lieber dem World Wide Web als den Informationen aus dem eigenen Umfeld (12 Prozent).

Sobald die Entscheidung getroffen ist, wo die Reise hinführen soll, wird auch meist online gebucht. Hierzulande sind es 2016 fast drei Viertel (69 Prozent) der Befragten, die ihren Urlaub im Internet buchen möchten, 4 Prozent mehr als 2015 (65 Prozent). Europaweit jedoch ist England Spitzenreiter (75 Prozent). Und auch in Italien gehen fast drei Viertel (72 Prozent) aller Befragten für die Urlaubsbuchung lieber online als ins nächste Reisebüro. Knapp dahinter folgt Spanien (71 Prozent). Frankreich landet mit 66 Prozent hinter Deutschland auf Platz fünf.

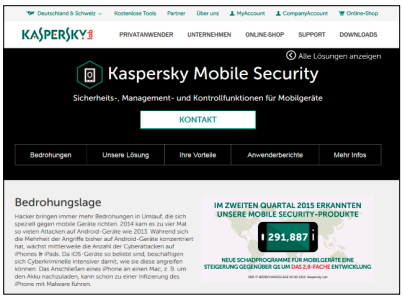
Die Umfrage wurde im Auftrag von Lastminute.de durch OnePoll durchgeführt.

www.lastminute.de

Kaspersky-Studie

Infizierung via USB-Ladevorgang

Wie sicher sind frei verfügbare Smartphone-Ladestationen an Flughäfen, in Bars oder im öffentlichen Nahverkehr? Diese und ähnliche Fragen haben die Experten von Kaspersky Lab im Rahmen einer Machbarkeitsstudie untersucht und sind zu dem Ergebnis gekommen, dass Smartphones während des Ladevorgangs per USB-Verbindung kompromittiert werden können.



Kaspersky-Studie offenbart Lücken für Heim- und Unternehmensanwender

Bei der Kaspersky-Studie wurden im ersten Schritt eine Reihe von Smartphones unter verschiedenen Android- und iOS-Betriebssystemen dahingehend untersucht, welche Daten das Gerät während des Ladevorgangs mit einem PC oder Mac extern preisgibt. Die Testergebnisse zeigen: Die mobilen Geräten offenbaren – abhängig von Gerät und Anbieter – dem Computer eine Reihe an Daten, wie Gerätename, Hersteller, Gerätetyp, Seriennummer, Firmware- und Betriebssysteminformationen, Dateisystem/Dateiliste sowie die elektronische Chip-ID.

Das Sicherheitsproblem: Smartphones – als ständiger Begleiter – werden so für Dritte interessant, die an der Sammlung solcher Daten interessiert sein könnten, um diese im Anschluss für sich zu nutzen.

»Die Sicherheitsrisiken sind offensichtlich: Nutzer können über die IDs ihrer Geräte verfolgt und das Mobiltelefon kann heimlich infiziert werden. Entscheidungsträger großer Unternehmen könnten so leicht zum Ziel professioneller Hacker werden«, sagt Alexey Komarov, Sicherheitsforscher bei Kaspersky Lab. »Die Hacker müssen nicht einmal hochqualifiziert sein, um eine solche Attacke auszuführen, denn alle erforderlichen Informationen sind im Internet leicht zu finden.«

Auf der Black-Hat-Konferenz 2014 wurde bereits gezeigt, dass man Smartphones mit einem Schädling infizieren kann, indem man es mit einer fingierten Ladestation verbindet.

Experten von Kaspersky Lab haben das Szenario reproduziert. Dazu reichten ein gewöhnlicher PC, ein Standard-Mikro-USB-Kabel sowie einige bestimmte Befehle. Damit konnte heimlich eine sogenannte Root-App auf einem Smartphone installiert werden. Das heißt, das Smartphone wurde ohne die Verwendung eines Schadprogramms kompromittiert.

www.kaspersky.com/de

Zahl des Monats

Während im letzten Quartal 2015 noch **29,6 Prozent** der Deutschen per Smartphone ins Internet gingen, lag der Anteil im ersten Quartal 2016 bei fast **32 Prozent**. Dagegen sind Desktop-Rechner und Laptops im Vergleich der klare Verlierer. Waren diese im letzten Quartal 2015 noch für **60 Prozent** der Deutschen die bevorzugten Endgeräte, so surfen in Deutschland 2016 nur noch knapp **57 Prozent** darüber. Einzig die Nutzung der Tablets bleibt mit 11 Prozent stabil. Quelle: www.webtrekk.com

Samsung-Studie

Technikkompetenz der Europäer

Eine Umfrage unter mehr als 10.000 europäischen Konsumenten offenbart großes Interesse an Technikthemen, aber auch Wissenslücken bei Fachbegriffen.

Neue Technologien wie Virtual-Reality-Brillen, fahrerlose Autos und intelligente Kühlschränke haben das Potenzial, beeindruckende neue Erlebnisse zu schaffen – und sowohl die Deutschen (64 Prozent) als auch die Europäer insgesamt (55 Prozent) finden neue Technik spannend. Dies ermittelte Samsung im Rahmen einer aktuellen, europaweiten Verbraucherumfrage, den Tech Habits 2016.

Fast die Hälfte der Befragten in Deutschland (46 Prozent)

Samsung-Studie: 46 Prozent der Deutschen nutzen mehr Technik als vor zwei Jahren

nutzt schon heute mehr Technik im Alltag als noch vor zwei Jahren. Doch die neue Technikwelt mit ihren vielen Fachwörtern ist auch komplex: 76 Prozent der Deutschen haben schon einmal vorgegeben, einen Begriff zu kennen, ohne ihn wirklich zu verstehen.

»Unsere neue Verbraucherstudie zum Thema Technik zeigt, dass sowohl die Deutschen als auch die Europäer neuen Technologien sehr positiv gegenüberstehen«, sagt Georg R. Rötzer, Vice President Corporate Marketing Samsung Electronics GmbH. »Doch die Ergebnisse offenbaren auch, dass Technikspräche für manche Verbraucher noch fremd ist und dass neue Technologien nicht nur smart, sondern leicht verständlich sein müssen. Wir als Hersteller fühlen uns verpflichtet, den Verbrauchern die Vorteile neuer Technologien aufzuzeigen. Deswegen investieren wir in Produkte, die smart, benutzerfreundlich, sicher und einfach zu verstehen sind. Zusätzlich bieten wir professionellen Kundenservice mit umfassenden Möglichkeiten der Kontaktaufnahme, von Social Media bis hin zum persönlichen Gespräch«, so Rötzer weiter.

www.samsung.com

Centrify

Neues Developer-Programm gestartet

Mit dem neuen Centrify Developer Program können Entwickler Single-sign-on, Multi-Faktor-Authentifizierung und Access Control vereinfachen. Digitale Identitäten und Optionen für IT-Sicherheit stehen jetzt über APIs zur Verfügung.

Als Teil des neuen Centrify Developer Programs stehen die Funktionen von Centrify für Digitale Identitäten und IT-Sicherheit auch über APIs zur Verfügung und nicht nur als ▶

Bitkom Research

Jedes zweite Unternehmen nutzt die Cloud

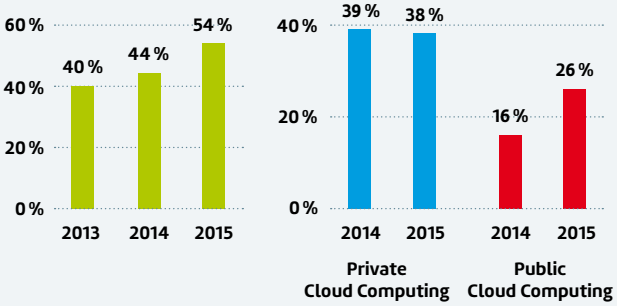
Zum ersten Mal hat im vergangenen Jahr eine Mehrheit der Unternehmen in Deutschland Cloud Computing eingesetzt. Das hat eine Umfrage von Bitkom Research im Auftrag der KPMG AG ergeben. Demnach nutzten 54 Prozent der Unternehmen im Jahr 2015 Cloud Computing. Im Jahr davor waren es erst 44 Prozent. Weitere 18 Prozent der Befragten planten oder diskutierten im vergangenen Jahr den Einsatz. »Cloud Computing ist eine Killer-Applikation der Digitalisierung«, sagte Dr. Axel Pols, Geschäftsführer von Bitkom Research, bei der Vorstellung der Studienergebnisse. »Die Technologie schafft enorme Effizienzgewinne und sie ist in der digitalen Wirtschaft sehr häufig die Basis neuer Geschäftsmodelle.« Der starke Anstieg der Nutzung ist laut Umfrage fast ausschließlich auf kleinere und mittlere Unternehmen zurückzuführen.

So stieg die Cloud-Nutzung in Unternehmen mit 100 bis 1999 Mitarbeitern um 7 Prozentpunkte auf 62 Prozent im Jahr 2015 und in Unternehmen mit 20 bis 99 Mitarbeitern sogar um 11 Punkte auf 52 Prozent. Bei Unternehmen ab 2000 Mitarbeitern legte die Nutzung auf vergleichsweise hohem Niveau nur um einen Punkt auf 69 Prozent zu.

Die am weitesten verbreitete Public-Cloud-Anwendung ist laut Umfrage Büro-Software. 43 Prozent der befragten Unternehmen nutzen zum Beispiel Textsysteme, Tabellenkalkulation oder Programme zur Erstellung von Präsentationen. 35 Prozent setzen Groupware ein, 34 Prozent branchenspezifische Anwendungen und 30 Prozent Software für die Organisation von Arbeitsgruppen (Collaboration Tools). Immerhin 29 Prozent nutzen spezielle Sicherheitsanwendungen unter dem Stichwort Security as a Service über das Internet.

www.bitkom.org

Nutzung von Cloud Computing in Unternehmen



Die Mehrheit der Unternehmen in Deutschland nutzt Cloud Computing.

web & mobile developer 8/2016 Quelle: Bitkom Research,KPMG

Veracode-Studie

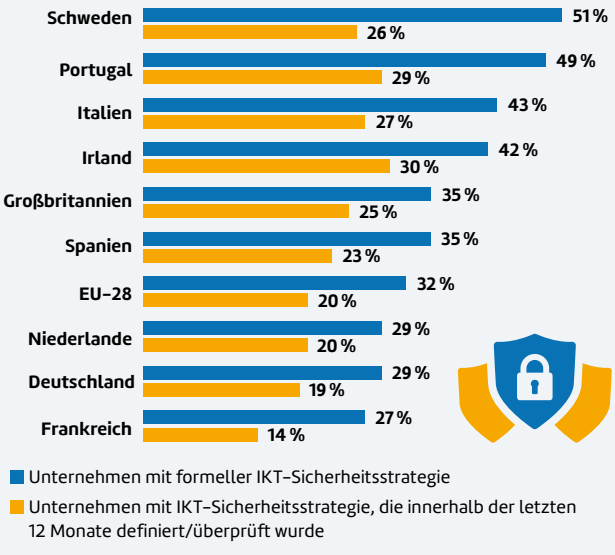
65 Milliarden Euro Schaden durch Cyber-Angriffe

Deutschen Unternehmen sind in den letzten fünf Jahren Schäden von insgesamt 65,2 Milliarden Euro durch Internetattacken entstanden. Das ergibt für die deutsche Wirtschaft eine jährliche Schadenssumme von rund 13 Milliarden Euro. Besonders stark betroffen ist die herstellende Industrie. Zu diesem Ergebnis kommt eine Studie des Center for Economics and Business Research (Cebr) im Auftrag des Spezialisten für Anwendungssicherheit Veracode. Vor allem Web- und Cloud-Anwendungen als Einfallstor bereiten Unternehmen demnach Sorge. Zudem untersucht die Studie, wer im Unternehmen im Fall eines Cyber-Sicherheitsvorfalls die Verantwortung tragen sollte.

Für die Studie wurden deutsche Unternehmen mit über 1000 Beschäftigten befragt. Im Schnitt wurde jedes befragte Unternehmen in den letzten fünf Jahren zweimal Opfer eines Cyber-Angriffs. Überdurchschnittlich oft traf es Firmen im Baugeschäft mit 2,7 und die Logistikunternehmen mit 2,5 Attacken aus dem Netz. Die Schäden durch diese Angriffe in den letzten fünf Jahren verteilen sich höchst unterschiedlich auf verschiedene Branchen: Die herstellende Industrie hat mit 27 Milliarden Euro Schaden die höchsten Schäden erlitten. Die Versorgungs-, Industrie- und Bergbaubranche hat Schäden von 9,2 Milliarden Euro zu beklagen. Die Baubranche verzeichnete mit 6,5 Milliarden Euro die dritthöchste Schadenssumme.

»Kein Unternehmen in Deutschland sollte Investitionen in die Cyber-Sicherheit vernachlässigen«, erklärt Markus Schaffrin, Sicherheitsexperte beim Branchenverband der Internetwirtschaft eco. »Die Studie zeigt, dass erfreulicherweise die meisten großen Unternehmen das erkannt haben und planen, ihre Ausgaben für IT-Sicherheit zu erhöhen. Die restlichen sollten sich fragen: Wie teuer ist es für ein Unternehmen, wenn es gehackt wird und Cyber-Kriminelle Produktionsanlagen lahmlegen, geistiges Eigentum oder private Daten entwenden?«
http://de.veracode.net

Anteil der Unternehmen mit IKT-Sicherheitsstrategie (2015)



web & mobile developer 8/2016 Quelle: Eurostat

eigenständige Produkte. Kunden und App-Entwickler profitieren von den benötigten Sicherheits-Features und können sie gleichzeitig anpassen, um ihre individuellen Anforderungen zu erfüllen. Kunden können die Lösungen von Centrify integrieren und das Look and Feel vorhandener IT-Umgebungen beibehalten. Entwickler können sich auf ihre spezifische Applikationsexpertise fokussieren und Centrifys APIs nutzen, um die Nutzer- und zugriffsbezogenen Aspekte ihrer Applikationen zu realisieren.

»Anderen Entwicklerprogrammen fehlen oft die anpassbaren, Identitäten-basierten Security APIs für Endanwender und privilegierte Anwender«, sagt Chris Webber, Security Strategist bei Centrify. »Mit dem Start des Centrify Developer Programs können Kunden und App-Entwickler unkompliziert auf unsere APIs und die dazugehörigen Dokumentationen zugreifen.«
http://developer.centrify.com

Shopware Community Day

Blick in die E-Commerce-Zukunft

1600 Händler, Partner, Entwickler und E-Commerce-Enthusiasten versammelten sich in Ahaus und informierten sich in rund 60 Fachvorträgen und bei 40 Ausstellern. Außerdem erlebten sie, wie Shopware-Vorstand Stefan Hamann in seiner Keynote alle Neuigkeiten rund um Shopware vorstellte.

Hamann präsentierte in seiner Keynote Shopwares Visionen der E-Commerce-Zukunft und skizzierte, wie der Software-Hersteller diesen Weg beschreitet. »Wir beschäftigen uns im Bereich Research und Development mit Dingen, die derzeit noch experimentell sind, aber später direkten Einzug ins Produkt finden. Dazu gehören etwa Shopping-Ansätze in der virtuellen Realität (VR), das Internet of Things sowie andere Themen aus dem Bereich Future of Retail. Diese Konzepte behandeln wir konsequent nach dem Design-Thinking-Ansatz und beziehen dabei insbesondere unsere Community stark mit ein, zum Beispiel über

Shopware-Vorstand Stefan Hamann auf dem Shopware Community Day

unser Shopware Insider Programm.« Weitere Foren für den Wissenstransfer aus der Community seien etwa regelmäßig stattfindende Think Tanks, externe Hackathlons sowie die transparente Entwicklung, wie Shopware sie auch über die soziale Coding-Plattform GitHub vorantreibt.

Während des Community Days stand vor allem der Open-Source-Gedanke im Vordergrund, der auch die Entwicklung der Version 5.2 begleitete. Stefan Hamann stellte die neuen Features vor, die sowohl Händlern und Entwicklern als auch Endkunden zugute kommen werden. Dazu gehören verbesserte Einkaufswelten, ein neues Adressmanagement oder ein überarbeitetes Plug-in-System. Ab Shopware 5.2 ist die Basisversion des Warenwirtschaftssystems Pickware direkt und kostenlos in die Professional und Professional Plus Edition integriert.
www.shopware.com

Jetzt kostenlos testen!

2 x gratis!

Praxiswissen für Entwickler!

Testen Sie jetzt 2 kostenlose Ausgaben und erhalten Sie exklusiven Zugang zu unserem Archiv.

webundmobile.de/probelesen

Software AG

webMethods DevOps-Edition vorgestellt

Die Software AG hat die webMethods DevOps-Edition angekündigt. Damit kann Software kontinuierlich entwickelt, getestet, integriert und mit einer nahe null liegenden Fehlerrate ausgeliefert werden.

Auf der operationalen Seite senkt das Produkt den Zeitaufwand für das Recovery und unterstützt eine enge Zusammenarbeit zwischen den Entwicklungsabteilungen und IT-Operations.

Dr. Wolfram Jost, CTO Software AG, führte aus: »Schneller als mit unserer Digital Business Platform kann man neue Geschäftsmodelle nicht umsetzen. Eine kontinuierliche Implementierung neuer Modelle

Dr. Wolfram Jost ist CTO der Software AG

erfordert, dass Entwickler die Schnittstellen zwischen den unterschiedlichen Systemen zügig anpassen können.«

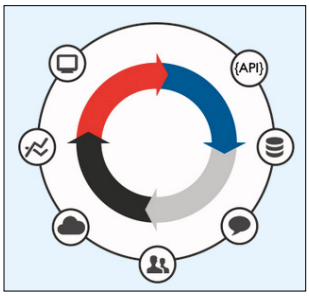
Das Highlight der Edition sind die Funktionen für unplugged und automatisierte Tests. Entwickler können Apps und neue Funktionen programmieren, ohne mit einem Netzwerk oder einem Server verbunden zu sein.

www.softwareag.com/de

Embarcadero Technologies

RAD Server: Backend für C++Builder- und Delphi-Anwendungen

Embarcadero Technologies hat mit RAD Server eine Backend-Plattform zum Entwickeln und Verteilen von Anwendungsdiensten für Delphi und C++Builder vorgestellt.



Entwickeln und Verteilen von Anwendungsdiensten für Delphi und C++Builder mit dem RAD Server

Die Lösung erspart Entwicklern das bisher notwendige Programmieren eigener Backend-Server und -Services. Es genügt, so der Hersteller, Delphi- oder C++Builder-Methoden in den RAD Server zu laden, um Backend-Code für Clients über verwaltete REST/JSON-Endpunkte bereitzustellen. Die in RAD Server eingebauten Anwendungs-Services und -Integrationen bieten die gängigsten Backend-Funktionen und Zugriff auf die wichtigsten externen Systeme. RAD Server sorgt für eine einfache Verteilung auf lokale Hosts oder Cloud-Dienste wie Amazon oder Microsoft Azure.

Aufgrund seiner Nutzerfreundlichkeit, Verarbeitungskapazität und Vielseitigkeit taugt der RAD Server auch als Backend-Plattform für ISVs (Independent Software Vendors), die paketierte, wiederverwendbare Lösungen programmieren, oder für Teams, die Anwendungen für interne Zwecke schreiben, verspricht Embarcadero. RAD Server bringt flexible Lizenzierungsoptionen mit, um auf die Bedürfnisse der verschiedensten Entwickler eingehen zu können. Es können beispielsweise Lizenzen pro Anwender erworben werden, oder Single- beziehungsweise Multi-Site-Lizenzen, die jeweils unbegrenzt viele Anwender pro Applikation erlauben.

RAD Server steht ab sofort unter https://www.embarcadero.com/products/rad-server zum Download bereit.

www.embarcadero.com

NativeScript 2.0

Native Anwendungen für Mobilgeräte

Eine höhere Produktivität, umfangreiche Optionen für mobile Entwickler und mehr Kosteneffizienz verspricht der Hersteller von NativeScript 2.0, das auch mit Angular 2 zusammenarbeitet.

Progress hat den Release 2.0 des Open-Source-Frameworks NativeScript vorgestellt. Entwickler sollen damit in die Lage versetzt werden, JavaScript zur Erstellung nativer mobiler Apps einzusetzen, die auf allen wichtigen mobilen Plattformen laufen.

Auch Anwender des Frameworks Angular 2 können mit NativeScript 2.0 native mobile Applikationen für iOS und Android anfertigen und dabei vorhandenen Code und ihr Know-how weiter nutzen.

NativeScript 2.0 soll auch mit Angular 2 zusammenarbeiten

AngularJS ist ein weit verbreitetes Open-Source-JavaScript-Framework für die Applikationsentwicklung, und in der neuesten Version Angular 2 RC lässt sich Angular auch außerhalb eines Browsers einsetzen. Durch die Integration von NativeScript mit Angular 2 sinken die Kosten für die Erstellung mobiler Applikationen, so der Hersteller, denn Entwickler können bereits vorhandene Funktionalitäten und ihre Erfahrungen bei der Webentwicklung in ihren mobilen Applikationen weiterverwenden. Zudem bedeutet dies für Angular-Entwickler, dass sie jetzt mit einem geringen Aufwand auch native mobile Applikationen erstellen können.

NativeScript von der Progress-Tochter Telerik unterscheidet sich von anderen Entwicklungsumgebungen, denn es produziert native Apps, die auch auf älteren Android-Geräten performant laufen. Gleichzeitig ist es möglich, Know-how und Programmcode für Web- und andere Plattformen gemeinsam zu nutzen.

Aufbauend auf den Erfahrungen beim Erstellen nativer Apps mit JavaScript bietet der Einsatz von Angular zusammen mit NativeScript die Möglichkeit, Programmcode für Web- und mobile Applikationen gemeinsam zu nutzen.

www.nativescript.org

EnterpriseDB

EDB startet DBM-Plattform

EnterpriseDB hat die EDB-Postgres-Plattform vorgestellt, eine integrierte Datenbankmanagement-Plattform auf Open-Source-Basis für den professionellen Unternehmenseinsatz.

Die neue Plattform soll ein breites Spektrum an Bereitstel-

lungstopologien ermöglichen. Sie integriert EDBs Postgres-Datenbank mit weiteren Datenmanagement-Lösungen und bietet ein spezialisiertes Partner-Ökosystem für neue, agilere Bereitstellungsmodelle. Sie integriert Postgres mit MongoDB und Hadoop und verbessert DevOps und Database-as-a-Service mit OpenStack und

Die EDB Postgres-Plattform bietet ein breites Spektrum an Bereitstellungstopologien

Multi-Cloud-Unterstützung, so der Hersteller.

Die Datenmanagement-Plattform auf Open-Source-Basis vereinigt alle für die Verwaltung von strukturierten und unstrukturierten Daten in einem föderierten Modell erforderlichen Komponenten. Die Plattform erlaubt die Integration verschiedener Datenbanken und unterstützt zugleich die Kombination unstrukturierter Daten von NoSQL-Lösungen mit transaktionalen Unternehmensdaten in strukturierten relationalen Systemen.

Interessenten haben die Wahl zwischen zwei Database-Serverversionen, dem EDB Postgres Standard Server, einer Open-Source-PostgreSQL-Datenbank, oder dem EDB Postgres Advanced Server, einer erweiterten Version von PostgreSQL mit ausgebauter Performance, Skalierbarkeit sowie Sicherheitsfunktionen und Kapazitäten für den professionellen Unternehmenseinsatz.

Zudem umfasst die Plattform eine Management-Suite, eine Migration- und eine Integration-Suite sowie Support und Services. ▶

Cloud

Die wachsende Bedeutung des Cloud Computing

Über 80 Prozent des weltweiten Datenverkehrs zwischen Rechenzentren weltweit wird noch vor dem Jahr 2020 aus der Cloud kommen. Mit dieser Prognose will eco – Verband der Internetwirtschaft e. V. auf die rasant wachsende Bedeutung des Cloud Computing hinweisen.

»Vier von fünf Datenzentren werden laut Studien schon 2019 Cloud-Daten verarbeiten«, sagt Andreas Weiss, Direktor der EuroCloud Deutschland_eco e. V. Grundlage bildet die Annahme, dass bis dahin rund 25 Milliarden Geräte mit Internetanschluss weltweit im Einsatz sind. Das erzeugte Datenvolumen wird laut Cisco Global Cloud Index auf mehr als 40 Zettabyte pro Monat geschätzt.

Grundlage für diese Einschätzungen bildet die Annahme, dass in den nächsten Jahren weite Teile der Wirtschaft ihre Datenspeicherung und -verarbeitung in die Cloud legen werden.

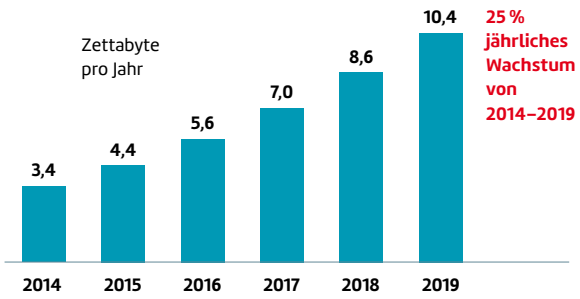
»Die Funktions- und vor allem die Kostenvorteile der Cloud-Services gegenüber firmeneigenen Rechenzentren sind derart hoch, dass Konzerne wie Mittelständler ihre bisherige Zurückhaltung gegenüber Cloud-Lösungen zügig ablegen werden«, sagt Andreas Weiss voraus.

Darüber hinaus wird sich nach Einschätzung von eco das Internet of Things (IoT) in den nächsten Jahren massiv ausbreiten und für heute noch kaum vorstellbare Datenströme sorgen. »Künftig wird jede Maschine, jedes Haushaltsgerät und so weiter mit dem Internet verbunden sein«, erklärt Dr. Béla Waldhauser, Leiter der Kompetenzgruppe Datacenter Infrastruktur im eco Verband. Allein die Digitalisierung der Kraftfahrzeuge und die Verbreitung von Mobile Payment als Standardzahlungsmittel werde enorme Datenmengen produzieren. »Wir reden nicht davon, dass das Auto eine Internetverbindung bereitstellt, wenn gelegentlich ein Mitfahrer surfen will, sondern davon, dass der Wagen im Millisekundentakt Informationen an den Hersteller und die Verkehrsinfrastruktur übermittelt, die Musik via Streaming ankommt und die Windschutzscheibe fortlaufend mit Virtual Reality aktualisiert wird«, verdeutlicht Dr. Béla Waldhauser. Er verweist auf Schätzungen aus dem Cisco VNI Complete Forecast Update 2015, die von mehr als 10 Milliarden Machine-to-Machine-Internetverbindungen noch vor dem Jahr 2020 ausgehen.

Als einen wesentlichen Treiber des Internet of Things nennt eco die Markenartikelindustrie. »Jede Zahnbürste, jeder Rasierer und jede Kaffeemaschine, die der Hersteller mit einem Internetanschluss und einer mobilen App versieht, adressiert eine neue Zielgruppe, die zudem gerne bereit ist, für die Innovation einen Aufpreis zu bezahlen«, skizziert Dr. Béla Waldhauser einen neuen Milliardenmarkt. Als strategisch noch wichtiger bezeichnet er die Erkenntnis, dass sich aus den von den Geräten übermittelten Daten neue Geschäftsmodelle entwickeln lassen, die unter Umständen noch größer als der Ursprungsmarkt sein können. »Wer die Zähne seiner Kunden zweimal täglich scannt, dem fällt vielleicht noch mehr ein, als nur Zahnbürsten zu verkaufen«, macht Dr. Béla Waldhauser seine Einschätzung anschaulich. Er weiß aus Gesprächen: »Viele Industrievorstände sind beeindruckt, wie es AirBnB und Uber gelingt, aus den gesammelten Daten über Big-Data- und Smart-Data-Analysen neue Geschäftsmodelle zu entwickeln, und betrachten das durchaus als Vorbild für ihre eigene Zukunft«. Waldhauser weiß auch: »Bei diesen Datenvolumina wird es eng in den Rechenzentren.«

www.eco.de

Bald 40 Zettabyte pro Monat



Der weltweite Datenverkehr zwischen Rechenzentren nimmt in den kommenden Jahren rasant zu.

web & mobile developer 8/2016 Quelle: Cisco Global Cloud Index, 2014–2019

IBM

Watson assistiert bald auch Sicherheitsexperten

Das kognitive System IBM Watson soll zukünftig IT-Sicherheitsexperten bei der Analyse und dem Monitoring von Cybergefahren unterstützen. Aktuell läuft ein Pilotprojekt mit acht US-Eliteunis.

In Kooperation mit acht US-amerikanischen Elite-Universitäten startet IBM Security ein Pilotprojekt, um das kognitive System IBM Watson fit für die Cyberabwehr zu machen. Dafür wird Watson von rund 200 Teilnehmern intensiv trainiert. Ziel ist es, IT-Sicherheitsexperten in Unternehmen bei Analyse, Auswertung sowie Monitoring von Cybergefahren zu unterstützen. Darüber hinaus soll Watson im Fall eines Angriffs auch Empfehlungen für die jeweils passende Sicherheitsstrategie geben. Ab Herbst 2016 werden IT-Studenten der am Pilotprojekt beteiligten Universitäten damit befasst sein, große, themenbezogene Datenmengen dem kognitiven System zuzuführen und es für kommende Aufgaben zu trainieren.

Analyse und Monitoring von Cybergefahren mit Hilfe von IBM Watson

Jüngste Studien belegen, dass Unternehmen im Durchschnitt täglich rund 200.000 Hinweise auf Sicherheitsvorfälle bekommen, die in den allermeisten Fällen zwar eher unkritisch sind, deren Bearbeitung aber etwa 21.000 Stunden pro Jahr beziehungsweise 1,3 Millionen Dollar kostet. Daher starten ab Herbst 2016 Studenten von acht renommierten US-amerikanischen Universitäten mit anerkannten Lehrstühlen für IT-Sicherheit ein Pilotprojekt mit IBM Security als Partner. Das kognitive System IBM Watson wird dazu von den etwa 200 Teilnehmern mit Wissen und Erkenntnissen rund um das Thema IT-Sicherheit gefüttert – unter anderem mit Informationen zu Malware, zu existierenden Cyberstrategien oder mit ganzen Datenbanken zur Historie von Cyberangriffen.

In diesem Hands-on-the-Job-Projekt werden weltweit erstmalig Studenten der Informationstechnologie praktische Erfahrung mit kognitiver IT-Sicherheit sammeln. Die Partneruniversitäten sind: die Polytechnische Universität Kaliforniens, Pomona; die Pennsylvania State University; das Massachusetts Institute of Technology; die New York University; die UMBC; die University of New Brunswick; die University of Ottawa und die University of Waterloo.

Ziel von IBM ist es, Watson mit bis zu 15.000 Sicherheitsdokumenten pro Monat zu füttern. Dazu gehören auch die Berichte der IBM X-Force-Teams, die seit über 20 Jahren zur Cybersicherheit forschen.

www.ibm.com

Die EDB-Postgres-Plattform verspricht schnellere Multi-Master-Replikation, die Bereitstellung privater Clouds auf OpenStack, mehr Leistung und Integration in verteilten Systemen und neue Datenadapter für die Integration von EDB Postgres mit anderen Datenmanagement-Lösungen, um hybride Datenplattformen zu erstellen.

Zu den neuesten Funktionen zählen der EDB Replication Server, Cloud Management, Hadoop, MongoDB und MySQL Datenadapter.

Dank des Partner-Ökosystems von EDB haben Kunden die Wahl zwischen lokaler Installation vor Ort, in der Cloud oder als Hybrid-Bereitstellungsmodell. In Sachen Hardware arbeitet EDB mit IBM, HP Enterprise und Dell zusammen an der Zertifizierung von EDB Postgres für High-Performance-Umgebungen.

www.enterprisedb.com

Neue Version von SAP HANA

Innovationen für das digitale Zeitalter

SAP hat die neueste Version der Plattform SAP HANA präsentiert – mit einer Vielzahl an Funktionen für zuverlässigere Unternehmensabläufe und größere Effizienz im Rechenzentrum.

Mit einem neuen Cloud-basierten Hybrid-Service sollen Kunden auch in ihrer lokalen Datenmanagement-Umgebung von der Flexibilität der Cloud und mehr Stabilität bei geschäftskritischen Workloads profitieren, während eine erweiterte Wartungsstrategie und ein neues Angebot für kleine und mittelständische Unternehmen den Weg in die digitale Geschäftswelt ebnen.

Mit der neuen Version der Plattform SAP HANA können sich Kunden durch gezielte Innovation einen Vorsprung vor ihren Mitbewerbern verschaffen und auch weiterhin von einer zuverlässigen Grundlage für ihre digitalen Geschäftsprozesse profitieren. Unter anderem stehen folgende neue Funktionen bereit:

Verarbeitung von Diagrammdaten: Ermöglicht Unternehmen die Visualisierung von Datenverbindungen, die komplexe Beziehungen zwischen Personen, Orten und Dingen aufzeigen. Das trägt zu einer besseren Betrugserkennung bei und hilft Unternehmen, durch die umfassende Analyse von Marktdaten neue Geschäftsmöglichkeiten zu erkunden.

Die neue Version der Plattform SAP HANA bietet erweiterte Cloud-Services und mehr Flexibilität

Erfassung und Wiedergabe: Durch die Erfassung von Systemlasten und deren Wiedergabe auf einem Zielsystem kann die IT-Abteilung neue Funktionen testen, verschiedene Upgrade-Optionen evaluieren und vorab untersuchen, wie sich geplante Änderungen auf das Produktivsystem auswirken. So können Administratoren neue Technologien ohne Ausfallzeiten und hohen Kostenaufwand testen.

Für die neue Version von SAP HANA werden auch zusätzliche Wartungsoptionen angeboten. IT-Organisationen können nun wahlweise ihre SAP-HANA-Umgebung für einen Zeitraum von bis zu drei Jahren warten oder sich Innovationen für die Plattform SAP HANA zweimal jährlich bereitstellen lassen.

www.sap.de

Jetzt kostenlos testen!

Das Fachmagazin für .NET-Entwickler

Testen Sie jetzt 2 kostenlose Ausgaben und erhalten Sie unseren exklusiven Newsletter gratis dazu.

www.dotnetpro.de/probeabo

QT QUICK CONTROLS 2

Alles auf Start

Mit Qt Quick Controls 2 lassen sich mobile Apps für Android, iOS und Windows 10 bauen, die gut aussehen und performant sind.

Im Rahmen meiner DSL-Artikelserie habe ich bereits einen kurzen Ausblick auf Qt 5.6 gegeben. Qt Quick Controls 2 wurden in Qt 5.6 als eine Tech Preview vorgestellt und sind jetzt mit Qt 5.7 offiziell verfügbar.

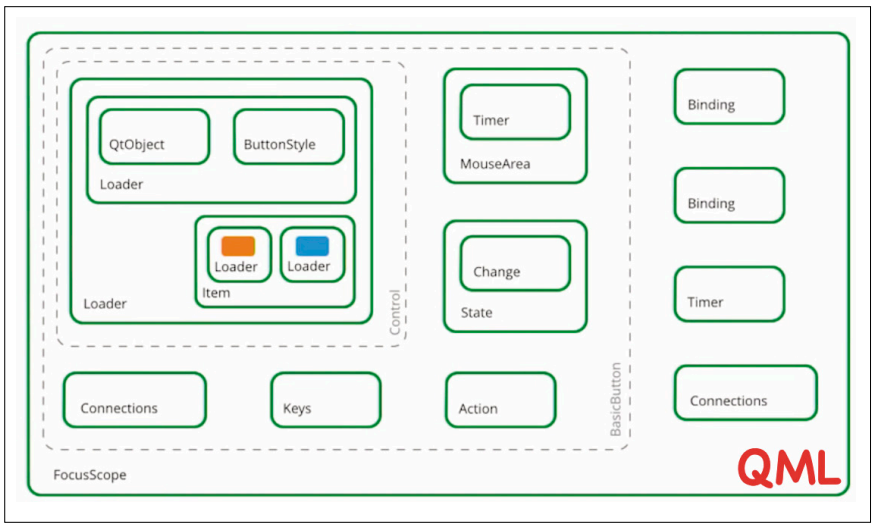
Leichtgewichtige Neuentwicklung

Qt gibt es ja seit mehr als 20 Jahren, und seitdem wurden verschiedene Wege verfolgt, mit Qt ansprechende Oberflächen zu entwickeln. Da gab es Qt Widgets, Qt Quick und Qt Quick Controls 2.

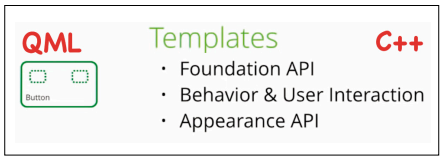
Für Qt-Newcomer ist das ein wenig verwirrend, da alle Technologien weiterhin im Einsatz sind. Qt wird auf unterschiedlichsten Plattformen eingesetzt: Desktop, Embedded und Mobil. Qt Widgets stellen native UI Controls insbesondere für Desktop-Anwendungen (OS X, Windows, Linux) zur Verfügung. Qt Quick wurde entwickelt, um UI Controls besser zu abstrahieren und cross-platform einsetzen zu können. Und Qt Quick schließlich basiert auf einer OpenGL-Implementierung. Es wurden dann auch mobile Apps mit Qt Quick gebaut. Diese sahen aber weder nativ aus, noch waren sie besonders performant. Es wurden auch keinerlei UI Controls zur Verfügung gestellt, die man von Android oder iOS gewohnt ist. Alles musste mit OpenGL selbst gebaut werden. Die Unterstützung von hochauflösenden Bildschirmen für mobile Devices war katastrophal.

All dies war der Auslöser, die Quick Controls komplett neu zu entwickeln, ohne auf bestehende Implementierungen aufzusetzen oder an irgendwelche APIs gebunden zu sein.

Dies war die Geburtsstunde der Qt Quick Controls 2, die jetzt auf Templates basieren und eine C++-Implementierung beinhalten. In QML erfolgt nur noch das Customizen. Diese Vorgehensweise ist ähnlich zu BlackBerry 10 Cascades, wo BlackBerry ein eigenes UI Framework rund um Qt und QML gebaut hat.



Qt Quick: Struktur eines Buttons bei Qt Quick (Bild 1)



Qt Quick Controls 2: Struktur eines Buttons bei Qt Quick Controls 2 (Bild 2)

Qt hat mit der Version 5.6 einen LTS-Release (Long Term Support) zur Verfügung gestellt, der Anwendern Sicherheit für die nächsten Jahre gibt. Dadurch wurde es möglich, bei QT 5.7 API-Änderungen vorzunehmen und auch die neuen Controls dort einzubauen.

Die bisherigen Quick Controls waren echte Schwergewichte, die alles, was eventuell auf irgendeiner der Plattformen benötigt wird, mit sich herumschleppen mussten – auch wenn es nicht zum Einsatz kam. Es ist ja ein großer Unterschied, ob ein UI Control auf einem Embedded-Device zum Einsatz kommt oder auf einer modernen Touch-Oberfläche wie Android oder iOS.

Die neuen Qt Quick Controls 2 enthalten nur noch das unbedingt Notwendige. Zusammen mit der C++-Implementierung können die neuen Controls wesentlich schneller instanziert werden. Was man dann an speziellen Funktionalitäten benötigt, fügt man einfach hinzu.

Bild 1 und Bild 2 zeigen das im Vergleich für einen Button, der nur noch etwa 10 Prozent der bisher benötigten Zeit zur Instanzierung benötigt. Mir gefallen die neuen C++/QML-Templates sehr gut, und sie erleichtern den Übergang von BlackBerry Cascades beziehungsweise die Entwicklung von Cross-Platform-Anwendungen für BlackBerry 10, Android, iOS und Windows10.

Unterschiedliche Bildschirmauflösungen

Vom technischen Gesichtspunkt her überzeugen die neuen Controls sofort. Wo sind die Unterschiede beim Einsatz für mobile Anwendungen? Zunächst einmal werden die unterschiedlichen Bildschirmauflösungen unter Android und iOS automatisch skaliert. Mit Qt 5.7 und Windows 10 habe ich bisher noch nicht experimentiert, daher bezieht sich der Artikel immer auf Android und iOS. Meine Testgeräte sind ein BlackBerry PRIV unter Android 6.0.1 und ein iPhone 6s unter iOS 9.3.

Mobile UI Controls für Android oder iOS muss man jetzt nicht mehr selbst bauen. Alles Wichtige gibt es, und was fehlt, kann per Customizing aus den zur Verfügung gestellten Templates schnell selbst gebaut werden.

Ich empfehle, einen Bugreport zu erstellen, wenn etwas vermisst wird. Das UI-Team ist sehr bemüht, Lücken zu schließen, und hat schon einige meiner Wünsche umgesetzt.

Es gibt nicht nur viele neue UI Controls, sondern auch neue Container, die das Entwicklerleben vereinfachen, sowie Navigations Controls, wie sie heute in mobilen Apps erwartet werden. Tabelle 1 bietet eine Übersicht.

Alle diese Controls sind derzeit in drei Styles verfügbar: Google Material, Windows Universal und Default – ein spezieller iOS Style wird folgen. Dunkles und helles Thema wird ebenfalls unterstützt.

Google Material Style

Im weiteren Verlauf des Artikels beschränke ich mich auf Googles Material Style, da man damit sehr gut auch Apps für die anderen Plattformen bauen kann – nicht nur für Android. Google macht es einem Entwickler auch leicht, Icons (basierend auf Schwarz oder Weiß) in den vielen erforderlichen Größen für die verschiedensten Bildschirmauflösungen einzusetzen, indem diese zum Download bereitgestellt werden.

Der Google Material Style Guide erklärt im Detail, wann welches UI Control in welcher Ausprägung zum Einsatz ▶

Tabelle 1: Qt Quick Controls 2

Qt Quick Controls 2 – Container	ApplicationWindow (Top Level Control), ButtonGroup, Container (ähnlich zum Cascades Container), Frame, GroupBox, Page (eine Fläche, die zusätzlich header und footer hat), Pane (eine für den gewählten Style beziehungsweise das Theme optimierte Fläche mit korrektem Hintergrund)
Qt Quick Controls 2 – Navigation	Drawer, StackView, SwipeView,TabBar, ToolBar
Qt Quick Controls 2 – Popup	Menu, Popup (Toasts/Dialoge), ToolTipp
Qt Quick Controls 2 – Buttons	Button, CheckBox, RadioButton, Switch, TabButton, ToolButton
Weitere Qt Quick Controls 2	BusyIndicator, ComboBox, Dial, Label, PageIndicator, ProgressBar, RangeSlider, ScrollBar, Slider, SpinBox, TextArea, TextField, Tumbler

kommt. In meinen ersten Beispielanwendungen für Qt 5.7 habe ich versucht, Anforderungen aus dem Style Guide exakt umzusetzen. Das hat – für eine erste Version der neuen Controls – erstaunlich gut geklappt. Die Apps stehen bei GitHub zum Download bereit.

Struktur einer einfachen App

Bild 3 zeigt einen Ausschnitt aus der *.pro*-Datei für die erste Beispiel-App, sowie die grobe Projektstruktur. Die *.pro*-Datei beschreibt die Anwendung – entspricht also in etwa der Manifest-Datei unter Android oder der *bardescriptor.xml* unter BlackBerry Cascades.

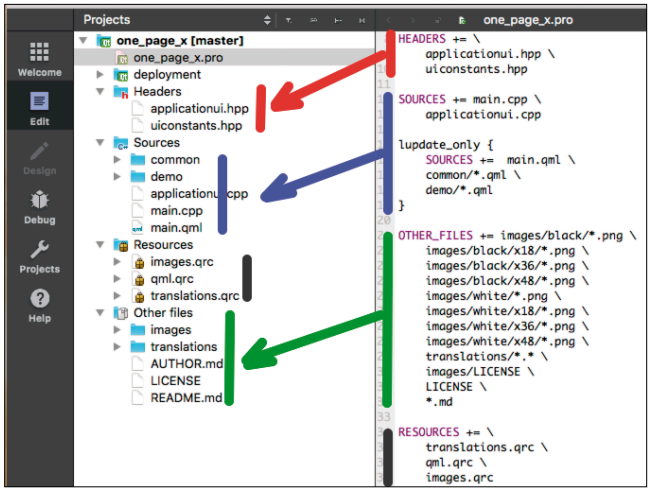
Kommt man von einer anderen IDE wie beispielsweise Eclipse, dann wird man ein Feature im Qt Creator – der Qt IDE – schmerzlich vermissen: die Abbildung des darunterliegenden Dateisystems. Kein Sync, kein Drag and Drop aus dem Finder ins Projekt et cetera. Es sind einige manuelle Schritte notwendig, um zumindest visuell eine ähnliche Ausgabe zu haben wie im Bild dargestellt.

Manches passiert automatisch: Beispielsweise werden beim Hinzufügen von C++-Dateien und -Headern diese in der *.pro* unter *SOURCES* beziehungsweise *HEADERS* eingefügt und sind dadurch auch links in der Projektübersicht dargestellt. QML-Dateien werden üblicherweise als Ressource in **.qrc*-Dateien verwaltet. Nach dem Anlegen einer *.qrc*-Datei erscheint diese auch unter *RESOURCES* in der *.pro* und ist links in der Übersicht zu sehen.

Keine klassische Ordnersicht

Fügen wir dann aber QML-Dateien der *.qrc* hinzu, dann ist es recht mühsam, sich da hindurchzuhangeln. Es gibt keine klassische Ordneransicht. Mit einem Trick bekommen wir die Inhalte aber dennoch vernünftig dargestellt. Um Strings zu übersetzen, müssen diese als *SOURCES* dem *LUPDATE*- und *LRELEASE*-Kommando zur Verfügung stehen. Andererseits soll aber der C++-Compiler diese nicht nutzen.

Daher der spezielle *lupdate_only*-Eintrag in der *.pro*-Datei. Dieser sorgt dafür, dass die Dateien fürs Übersetzen gefunden werden, und sie erscheinen gleichzeitig links in der Übersicht unter *SOURCES*. Durch die Verwendung von Wildcards müssen wir nur die Ordnerstruktur in der *.pro* definieren. Bei Bildern und Übersetzungsdateien bekommen wir diese links angezeigt, indem wir die Struktur unter *OTHER_FILES* in der *.pro* definieren.



Projektstruktur und *.pro*-Datei im Vergleich (Bild 3)

Manuelle Schritte erforderlich

Das Übersetzen von Strings erfolgt mit der Anwendung Qt Linguist zwar komfortabel, aber die Kommandos *LUPDATE* zum Extrahieren der Texte und *LRELEASE* zum Kompilieren der Übersetzungen werden nicht automatisch im Build-Prozess ausgeführt.

Auch an dieser Stelle sind ein paar manuelle Schritte notwendig – beziehungsweise sehr spezielle Eintragungen in der *.pro*-Datei.

Beim Starten der App wird zuerst die *main.cpp* ausgeführt (Listing 1). Wie man sehen kann, schaltet die *main.cpp* das *HIGH DPI SCALING* an, setzt den Material Style und startet den QTranslator, der für die Übersetzungen zuständig ist. Außerdem ermöglicht die *main.cpp* den Zugriff auf die Klasse *ApplicationUI* unter dem Namen *myApp*. Die *ApplicationUI*-Klasse stellt einige Methoden zur Verfügung, die aus dem UI heraus aufgerufen werden können:

```
Q_INVOKABLE
QStringList swapThemePalette();

Q_INVOKABLE
```

Listing 1: main.cpp

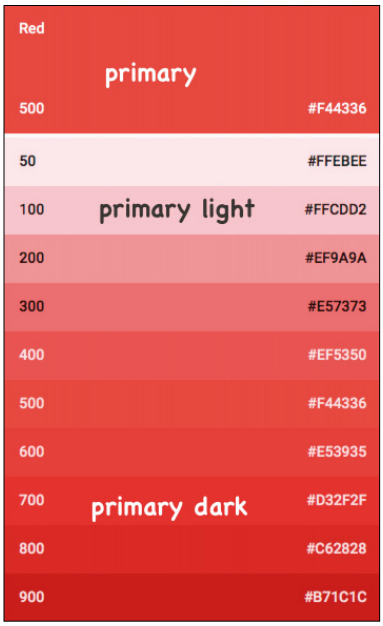
```
int main(int argc, char *argv[])
{
  QGuiApplication::setAttribute
  (Qt::AA_EnableHighDpiScaling);
  qputenv("QT_QUICK_CONTROLS_STYLE", "material");
  QGuiApplication app(argc, argv);
  QTranslator translator;
  if (translator.load(QLocale(), QLatin1String
  ("one_page_x"), QLatin1String("_"),
  QLatin1String(":/translations"))) {
    app.installTranslator(&translator);
  } else {
    qDebug() << "cannot load translator " <<
    QLocale::system().name() << " check content
    of translations.qrc";
  }
  ApplicationUI appui;
  QQmlApplicationEngine engine;
  QQmlContext* context = engine.rootContext();
  context->setContextProperty("myApp", &appui);
  engine.load(QUrl(QStringLiteral
  ("qrc:/main.qml")));
  return app.exec();
}
```

```
QStringList defaultThemePalette();

Q_INVOKABLE
QStringList primaryPalette(const int paletteIndex);

Q_INVOKABLE
QStringList accentPalette(const int paletteIndex);

Q_INVOKABLE
QStringList defaultPrimaryPalette();

Q_INVOKABLE
QStringList defaultAccentPalette();
```

Alle Methoden, die als *Q_INVOKABLE* markiert sind, können vom UI (QML) aus aufgerufen werden. Eine *QStringList* wird dabei automatisch auf ein JavaScript-Array gemappt:

```
ApplicationWindow {
  ...
  property variant primaryPalette:
  myApp.defaultPrimaryPalette()
  property color primaryLightColor:
  primaryPalette[0]
  property color primaryColor:
  primaryPalette[1]
  property color primaryDarkColor:
  primaryPalette[2]
  ...
```

Diese erste Beispielanwendung fungiert hauptsächlich als Playground für Google-Material-Farben, Fonts und mehr. Um darauf ohne große Umstände zugreifen zu können, habe ich ein paar Constants in C++ definiert, auf die ich dann aus QML zugreifen kann.



Farbschattierungen: Die Farbe *Red* aus dem Google Material Style Guide (Bild 4)

Vordefinierte Farben

Der Google Material Design Guide stellt eine Palette an vordefinierten Farben zur Verfügung. Qt 5.7 nutzt diese Farbpalette und empfiehlt ebenfalls, diese vordefinierten Farben für die *primaryColor* und die *accentColor* einzusetzen.

Sieht man sich die ganzen Beispiele im Design Guide an, wird man schnell feststellen, dass es auch oft genug notwendig ist, eine etwas hellere oder dunklere Variante der *primaryColor* zu nutzen. Daher habe ich in meine Paletten neben der 500er-Farbe auch 100 für eine hellere Version und 700 für eine dunklere Version eingetragen. Bild 4 zeigt als Beispiel die Farbe *Red* aus dem Google Material Style Guide.

Setze ich in Qt 5.7 die *Material.primaryColor*, dann wird diese beispielsweise in der ToolBar automatisch als Background genommen, und abhängig von der Farbe weiß Qt dann auch, ob die Vordergrundfarbe Weiß oder Schwarz ist. Dem Bild kann man entnehmen, dass die Vordergrundfarbe bei Verwendung der helleren oder dunkleren Primärfarbe unterschiedlich sein kann.

Es wird ja nicht nur Text auf der Primär- oder Akzentfarbe platziert, sondern auch Icons. Je nach Hintergrundfarbe/Theme empfiehlt Google, dabei ein weißes oder schwarzes Icon zu nutzen und gegebenenfalls die Opacity zu verändern. Der Download der Google Material Icons liefert alle Icons sowohl in Schwarz als auch in Weiß. Um mir all dies zu vereinfachen, habe ich für alle vordefinierten Farben entsprechende Werte in meinen Paletten hinterlegt:

- primaryLightColor (shade 100)
- primaryColor (shade 500)
- primaryDarkColor (shade 700)
- textOnPrimaryLight (white or black text color)
- textOnPrimary (white or black text color)
- textOnPrimaryDark (white or black text color)
- iconOnPrimaryLightFolder (images/dark or images/white)
- iconOnPrimaryFolder (images/dark or images/white)
- iconOnPrimaryDarkFolder (images/dark or images/white)

Für *Material.Red* sieht das dann folgendermaßen aus:

```
static const QStringList materialRed
{
  "#FFCDD2", "#F44336", "#D32F2F",
  "#000000", "#FFFFFF", "#FFFFFF",
  "black", "white", "white"
};
```

Jetzt ist es auch einfach, den QML-Code weiter oben zu verstehen: *myApp.defaultPrimaryPalette()* gibt ein Array aus C++ an das UI (QML) und stellt beispielsweise *primaryPalette[2]* die *primaryDarkColor* (#D32F2F) bereit. Diese Paletten können sowohl für die Primär- als auch für die Akzentfarbe genutzt werden. Der Sourcecode liegt bei GitHub bereit, kann also von jedem genutzt werden.

Textfarbe automatisch ableiten

Wie wir bereits wissen, unterstützt Qt 5.7 auch das dunkle und helle Theme und leitet davon automatisch die Textfarbe ab, die beispielsweise in einem Label genutzt wird. Aber auch beim Einsatz des dunklen oder hellen Themes gibt es spezielle Fälle, wo Farben oder Deckkraft (Opacity) sich abhängig vom Theme verändern. Daher habe ich auch dort entsprechende Paletten gebaut, die folgende Werte enthalten:

- *dividerColor* – siehe Google Material Dividers specs,
- *cardAndDialogBackground* – siehe Google Material Color Themes,
- *primaryTextOpacity* – siehe Google Material Text and Background Colors, ▶

- *secondaryTextOpacity* – siehe Google Material Text and Background Colors,
- *dividerOpacity* – siehe Google Material Dividers specs,
- *iconActiveOpacity* – siehe Google Material Icons style,
- *iconInactiveOpacity* – siehe Google Material Icons style,
- *iconFolder* – legt fest, ob die Icons vom Ordner *images/black* oder */white* kommen sollen,
- *isDark* – boolean on dunkles Thema.

Die beiden Paletten sehen folgendermaßen aus:

```
static const QStringList darkPalette {
  "#FFFFFF", "#424242", "1.0", "0.70", "0.12", "1.0",
  "0.3", "white", "1"
};
static const QStringList lightPalette {
  "#000000", "#FFFFFF", "0.87", "0.54", "0.12", "0.54",
  "0.26", "black", "0"
};
```

Jetzt ist es wirklich einfach, jeweils die korrekte Farbe und Deckkraft zu nutzen und zu wissen, ob ein weißes oder schwarzes Icon als Grundlage gewählt werden muss.

Eine Qt-Quick-Controls-2-App wird im UI als ein ApplicationWindow dargestellt. Dieses ApplicationWindow nimmt den ganzen zur Verfügung stehenden Platz ein, das heißt, es muss keine *width*- oder *height*-Angabe erfolgen, wenn die App nur auf mobilen Plattformen läuft.

Die erste einfache Beispielanwendung besteht nur aus einer langen scrollbaren Seite. Die grobgranulare Struktur:

```
import QtQuick 2.6
import QtQuick.Layouts 1.3
import QtQuick.Controls 2.0
import QtQuick.Controls.Material 2.0
import QtGraphicalEffects 1.0
import "common"
import "demo"
ApplicationWindow {
  visible: true
  // properties
  header: {}
  // optional footer {}
  Flickable {}
  // functions
  Popup {}
}
```

Bei den Import-Statements fallen zwei auf: *common* und *demo*. Beide sind Ordner, die QML-Dateien enthalten:

- *common:* QML-Dateien, die allgemeiner Art sind und meist in mehreren Apps zum Einsatz kommen.
- *demo:* QML-Dateien, die nur demonstrieren sollen, wie man solche Strukturen schafft.

Es ist immer eine gute Idee, die QML-Dateien zu strukturieren, dann ist die App übersichtlicher und es ist leichter verständlich, in welchem Kontext eine QML-Datei genutzt wird. Nicht vergessen sollte man, *visible* auf *true* zu setzen, sonst bekommt man nur einen leeren Screen.

Eindeutige Bezeichner für Properties

Properties, die anwendungsweit gelten, können am ApplicationWindow definiert werden und stehen dann allen anderen Controls zur Verfügung. Wenn man darauf achtet, dass die Properties eindeutige Bezeichner bekommen, kann auf ein Präfix verzichtet werden. Es reicht also *primaryLightColor* anstelle von *appWindow.primaryLightColor*. Listing 2 zeigt einige der Properties.

Neben den Properties befinden sich am ApplicationWindow auch anwendungsweit zur Verfügung stehende Funktionen – wie die folgende zum Ändern der Paletten:

```
function switchPrimaryPalette(paletteIndex) {
  primaryPalette = myApp.primaryPalette(paletteIndex)
}
function switchAccentPalette(paletteIndex) {
  accentPalette = myApp.accentPalette(paletteIndex)
}
```

Ein ApplicationWindow kann einen *header* und einen *footer* bekommen. Diese werden – wenn definiert – immer in der

Listing 2: Properties

```
// primary and accent properties:
property variant primaryPalette:
myApp.defaultPrimaryPalette()
property color primaryLightColor: primaryPalette[0]
property color primaryColor: primaryPalette[1]
property color primaryDarkColor: primaryPalette[2]
property color textOnPrimaryLight: primaryPalette[3]
...
Material.primary: primaryColor
Material.accent: accentColor
...
// theme Dark vs Light properties:
property variant themePalette:
myApp.defaultThemePalette()
property color dividerColor: themePalette[0]
property color cardAndDialogBackground:
themePalette[1]
property real primaryTextOpacity: themePalette[2]
...
property int isDarkTheme: themePalette[8]
onIsDarkThemeChanged: {
  if(isDarkTheme == 1) {
    Material.theme = Material.Dark
  } else {
    Material.theme = Material.Light
  }
}
```

Listing 3: Customized ToolBar

```
ToolBar {
 id: titleToolBar
 property alias text: titleLabel.text

 RowLayout {
  focus: false
  spacing: 6
  anchors.fill: parent
  LabelTitle {
   id: titleLabel
   text: ""
   leftPadding: 16
   elide: Label.ElideRight
   horizontalAlignment: Qt.AlignHCenter
   verticalAlignment: Qt.AlignVCenter
   color: textOnPrimary
  }
  ToolButton {
   Image {
    id: buttonImage
    anchors.centerIn: parent
    source: "qrc:/images/"+iconOnPrimaryFolder+
    "/more_vert.png"
   }
   onClicked: {
    optionsMenu.open()
   }
   Menu {
    id: optionsMenu
    x: parent.width - width
    transformOrigin: Menu.TopRight
    MenuItem {
     text: isDarkTheme? qsTr("Light Theme") :
     qsTr("Dark Theme")
     onTriggered: {
      themePalette = myApp.swapThemePalette()
     }
    }
    MenuItem {
     text: headlineColoredPrimary? qsTr
     ("Headline Accent Color") :
     qsTr("Headline Primary Color")
     onTriggered: {
      headlineColoredPrimary = !headlineColoredPrimary
     }
    }
   } // end optionsMenu
  } // end ToolButton
 } // end RowLayout
} // end ToolBar
```

App angezeigt. Wir nutzen hier nur einen *header* für die Titelzeile:

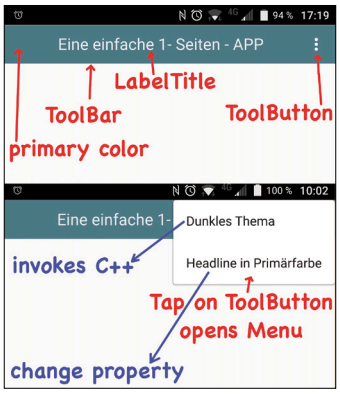
```
header: SimpleTextTitle {
  text: qsTr("A simple 1 - Page App")
}
```

Der Header ist eine customized ToolBar unter */common/SimpleTextTitle* (Listing 3).

Der *header* ist also in diesem Falle eine ToolBar – eines der neuen Qt Quick Controls 2. Eine ToolBar kann als *header* oder *footer* platziert werden. Beim Einsatz unter Material ist die *ToolBar.Header*-Position voreingestellt. Der Hintergrund einer ToolBar ist *Material.primaryColor*. Um Controls in der ToolBar zu platzieren, nutzt man am besten ein RowLayout. Unser RowLayout enthält ein Label als Titeltext und einen ToolButton rechts.

Image als ToolButton

Der ToolButton ist in diesem Fall ein Image. Zu beachten ist, wie die *source* unter Verwendung des *iconOnPrimaryFolder* konstruiert wird. *iconOnPrimaryFolder* ist eine der Properties unseres Root-Objekts *(ApplicationWindow)*. Der Ordnername ist *black* oder *white* – abhängig von der gewählten Primärfarbe. Bild 5 erläutert die Toolbar – einmal mit geschlossenem und einmal mit geöffnetem Menü.



Toolbar mit geschlossenem und geöffnetem Menü (Bild 5)

Es ist eine gute Idee, als äußeren Rahmen um den Inhalt ein Flickable zu packen. Nur dann kann der Inhalt nach oben oder unten über den Rand gezogen werden und springt dann zurück. Folgendes ist innerhalb des Flickable platziert:

```
Flickable {
  id: flickable
  contentHeight: root.implicitHeight
                 anchors.fill: parent
                 Pane {
                   id: root
                   anchors.fill: parent
                   ColumnLayout {
                     anchors.right: parent.right
                     anchors.left: parent.left
                     // ... controls ...
                   } // col layout
                 } // root
                 ScrollIndicator.vertical:
                 ScrollIndicator { }
} // flickable
```

Es war nicht einfach, herauszutüfteln, wie man die *contentHeight* des Flickable kor- ▶

rekt berechnet. Hier sind die Regeln:

- Ein Flickable muss die Höhe des Inhalts kennen *(contentHeight)*.
- *contentHeight* errechnet sich aus der impliziten Höhe des inneren Controls – in diesem Fall eine Pane.
- Das innere Control wiederum muss seinen *parent* – das Flickable – komplett ausfüllen: *anchors.fill: parent.*

Und nicht vergessen, den ScrollIndicator hinzuzufügen – sonst kann der Inhalt nicht gescrollt werden.

Container Controls

Das innere Control ist eines der neuen Container Controls: eine Pane. Das ist eine beliebige Fläche, die das aktuelle Theme und den Material Style kennt und daher immer automatisch den richtigen Background/Foreground zugewiesen bekommt.

Wer aus Gewohnheit anstelle einer Pane ein Rectangle nimmt und dann das Theme zwischen hell und dunkel wechselt, bekommt nur bei der Pane den richtigen Hintergrund.

Listing 4: Weitere Properties

```
// font sizes -
// defaults from Google Material Design Guide

property int fontSizeDisplay4: 112
property int fontSizeDisplay3: 56
property int fontSizeDisplay2: 45
property int fontSizeDisplay1: 34
property int fontSizeHeadline: 24
property int fontSizeTitle: 20
property int fontSizeSubheading: 16
property int fontSizeBodyAndButton: 14

// is Default property int fontSizeCaption: 12
// fonts are grouped into primary and secondary with
// different Opacity to make it easier to get
// the right property, here‘s the opacity per size:

property real opacityDisplay4: secondaryTextOpacity
property real opacityDisplay3: secondaryTextOpacity
property real opacityDisplay2: secondaryTextOpacity
property real opacityDisplay1: secondaryTextOpacity
property real opacityHeadline: primaryTextOpacity
property real opacityTitle: primaryTextOpacity
property real opacitySubheading: primaryTextOpacity

// body can be both: primary or secondary text

property real opacityBodyAndButton:
primaryTextOpacity
property real opacityBodySecondary:
secondaryTextOpacity
property real opacityCaption: secondaryTextOpacity
```

Achtung: Eine Pane hat ein Default-Padding. Verschachtelt man mehrere Panes und sollen die enthaltenen Controls ausgerichtet sein, dann muss bei den inneren Panes *padding* auf *0* gesetzt werden.

Werden innerhalb einer Pane die Controls Reihe für Reihe untereinander platziert, sollte ein ColumnLayout gewählt werden, das links und rechts verankert ist – also die ganze Breite einnimmt – und das sich dann beim Drehen des Geräts resized. Enthält nun eine dieser Reihen im ColumnLayout mehr als ein Control, dann sollten diese wiederum in ein RowLayout gepackt werden.

Cascades StackLayout

Vergleicht man das mit BlackBerry Cascades, dann entspricht ein ColumnLayout einem Cascades StackLayout mit Orientierung *TopToBottom* und ein RowLayout einem StackLayout *LeftToRight*.

Listing 5: Floating Action Button

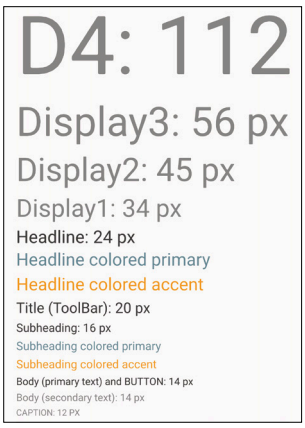
```
Button {
  id: button
  // image should be 24x24
  property alias imageSource: contentImage.source
  // default: primaryColor
  property alias backgroundColor:
  buttonBackground.color
  property bool showShadow: true
  contentItem:
    Item {
    implicitHeight: 24
    implicitWidth: 24
    Image {
      id: contentImage
      anchors.centerIn: parent
    }
  }
  background:
    Rectangle {
    id: buttonBackground
    implicitWidth: 56
    implicitHeight: 56
    color: primaryColor
    radius: width / 2
    opacity: button.pressed ? 0.75 : 1.0
    layer.enabled: button.showShadow
    layer.effect: DropShadow {
      verticalOffset: 3
      horizontalOffset: 1
      color: dropShadow
      samples: button.pressed ? 20 : 10
      spread: 0.5
    }
  }
}
```

Der Google Material Design Guide beschreibt sehr detailliert Fonts und verschiedene voreingestellte Größen, Farbe und Deckkraft unter dem Kapitel *Typography*. Ich habe versucht, all die verschiedenen Größen abzubilden. Um das zu vereinfachen, gibt es ein paar weitere Properties (Listing 4).

Für die verschiedenen Varianten habe ich unter */common* zugehörige Label definiert. Hier ein CaptionLabel unter *Common/LabelCaption.qml*:

```
Label {
  Layout.fillWidth: true
  font.pixelSize: fontSizeCaption
  opacity: opacityCaption
  font.capitalization: Font.AllUppercase
}
```

Bild 6 gibt einen kleinen Eindruck davon, wie das dann aussehen kann.



◀ **Fonts** und Customized Label (Bild 6)

Customized: Floating Action Button (Bild 7)

Floating Action Button (FAB)

Ein spezieller Button, der für Android-Material-Anwendungen wichtig ist und der mir in den bereitgestellten Controls gefehlt hat, war der Floating Action Button. Den habe ich mir dann schnell selbst erstellt. Listing 5 zeigt den Code. Der FAB basiert auf einem Button, der innen als *contentItem* ein Image hat, und dazu einen runden, farbigen Background. Unterhalb des Buttons kann noch ein Schatten dargestellt werden. Außer dem FAB gibt es noch einen Mini FAB in den Material Style Guides. Beides habe ich umgesetzt. Der Aufruf eines solchen FAB-Buttons ist dann einfach:

```
FloatingActionButton {
  imageSource: "qrc:/images/"+
  iconOnPrimaryLightFolder+"/person.png"
  backgroundColor: primaryLightColor
}
FloatingActionButton {
  imageSource: "qrc:/images/"+iconOnPrimaryFolder+
  "/person.png"
}
```

Bild 7 zeigt, dass die Floating Action Buttons wie in einer Material App erwartet aussehen.

Fazit

Mit wenig Aufwand kann man jetzt mit Qt 5.7 Anwendungen bauen, die unter Android und iOS gut aussehen und performant sind.

Für einen Newcomer ist es aber nicht einfach, sich im Qt-Universum zurechtzufinden, da es Qt für unzählige Plattformen gibt. Dieser Artikel soll helfen, einen schnellen Einstieg zu bekommen. In einem weiteren Artikel sehen wir uns dann an, wie mit *StackView*, *SwipeView* und *Drawer* durch eine Qt-5.7-Material-Style-App navigiert werden kann. ■

Links zum Thema

- Qt Quick Controls
  http://doc-snapshots.qt.io/qt5-5.7/qtquickcontrols2-index.html
- Qt Quick Container
  http://doc-snapshots.qt.io/qt5-5.7/qtquickcontrols2-containers.html
- Qt Quick 2 Navigation
  http://doc-snapshots.qt.io/qt5-5.7/qtquickcontrols2-navigation.html
- Google Material Style
  www.google.com/design/spec/material-design/introduction.html
- Qt Material Style
  http://doc-snapshots.qt.io/qt5-5.7/qtquickcontrols2-material.html
- Google Material Icons Download
  http://design.google.com/icons
- Erste Material Style Beispiel-App
  http://github.com/ekke/one_page_x
- Translations (i18n) mit Qt 5.7
  http://appbus.wordpress.com/2016/04/28/howto-translations-i18n
- Qt High DPI Dokumentation
  http://doc.qt.io/qt-5/highdpi.html

Ekkehard Gentz
ist Autor, Trainer und Speaker auf Konferenzen. Er entwickelt als Independent Software Architect mobile Anwendungen für internationale Kunden, ist BlackBerry Elite Member und bloggt unter:
http://ekkes-corner.org

NODE.JS MANAGEN MIT PM2

Das Ausloggen überleben

PM2 ist ein Prozess- und Deployment-Manager für Node.js mit vielen Features und einem ausgeklügelten Modulsystem für Erweiterungen.

Loggt man sich aus einem Benutzerkonto aus, dann werden üblicherweise alle darin noch laufenden Programme beendet. Wer Node.js für Serveranwendungen oder Microservices benutzt, steht schnell vor dem Problem, mit welchen Maßnahmen man die durch das Node-Executable gestarteten Prozesse in den Hintergrund legen kann, damit sie das Ausloggen überleben.

Sollen Dienste gar für viele Nutzer ausgelegt sein, kann es notwendig werden, mit einer Clustering-Strategie zu arbeiten. Es müssen dann mehrere zusammengehörige Prozesse verwaltet werden.

Beim Start und Neustart des Dienstes muss sichergestellt werden, dass alle Prozesse erfolgreich gestartet oder beendet werden. In Fehlersituationen möchte man Dienste automatisch neu starten lassen; jedoch höchstens mit einer begrenzten Anzahl an Versuchen – sonst droht eine Endlosschleife aus Dienst-Neustarts.

Es gibt dafür einige unterschiedliche Lösungen. Die wohl bekannteste ist das treffend benannte Skript forever von NodeJitsu. Die meisten Node.js-Tutorials zeigen, wie man mit forever einen Dienst starten und am Laufen halten kann. Doch dem beliebten NPM-Paket fehlen einige der oben erwähnten Features, die ein Prozessmanager insbesondere im Produktivbetrieb brauchen kann. Der Process Manager PM2 ist gerade deswegen als Ersatz für forever entworfen worden.

Installation von PM2 mittels NPM

PM2 wird per Node Package Manager (NPM) installiert. Dem Befehl können diverse Optionen mit auf den Weg gegeben werden:

```
$ npm install -g pm2
```

Mit der Option *-g* wird sichergestellt, dass das Paket global installiert wird und das Kommandozeilenskript *pm2* überall zur Verfügung steht.

Das Kommandozeilenskript bietet verschiedene Befehle. Einer davon ist *start*, mit dem man Anwendungen per PM2 starten kann. Eine Node.js-Anwendung startet man also beispielsweise folgendermaßen:

```
$ pm2 start app.js
```

Die Anwendung wird als Hintergrundprozess gestartet und von PM2 überwacht. Loggt man sich aus, läuft die Anwendung weiter. Eine sehr sinnvolle Zusatzoption ist *-n*:

```
$ pm2 start api.js -n webapi

[PM2] start process id 5
[PM2] Process successfully started
```

Damit kann man der Anwendung einen eigenständigen, leichter verständlichen Namen geben. Alle gerade bei PM2 registrierten Hintergrundprozesse kann man mit dem Befehl *list* auflisten:

```
$ pm2 list
```

PM2 lässt sich gut in andere Programme integrieren. Die Ergebnisse von *list* können deswegen mit dem Befehlt *jlist* auch als JSON abgerufen werden. Mit dem Befehl *stop* kann man laufende Prozesse stoppen:

```
$ pm2 stop all

[PM2] Stopping all
[PM2] stopProcessId process id 1
[PM2] stopProcessId process id 5
```

Statt *all* kann man auch die numerische Prozess-ID der Anwendung gezielt übergeben:

```
$ pm2 stop 5

[PM2] stopProcessId process id 5
```

Vergibt man Namen, kann man die Anwendungen auch damit ansteuern:

```
$ pm2 stop webapi
```

Verwendet man statt *stop* den Befehl *restart*, werden die Anwendungen zuerst gestoppt und danach gleich wieder gestartet.

Eine Besonderheit stellt der Befehl *reload* dar: Anwendungen, die bestimmten Kriterien entsprechen, können damit mit einer Null-Sekunden-Downtime neu gestartet werden. Dazu müssen diese Anwendungen HTTP/HTTPS/Socket-Netzwerkverbindungen nutzen und im PM2-Cluster-Modus gestartet werden.

Damit PM2 auch nach einem Neustart des Servers alle Anwendungen starten kann, müssen entsprechende Startup-

Skripts am Betriebssystem registriert werden. Das erfolgt mit dem PM2-Befehl *startup*:

```
$ pm2 startup
```

PM2 versucht automatisch, die Betriebsystemplattform zu ermitteln und die dazu passenden Startup-Skripts zu generieren. SystemV-init-Skripts, OpenRC (Gentoo), SystemD und LaunchAgents (Mac OS X) werden unterstützt.

Zusatzbibliothek für Windows-Nutzer

Auch wenn PM2 unter Windows betrieben werden kann, wurde der *startup*-Befehl noch nicht implementiert. Die Dokumentation schlägt Nutzern von Windows die Zusatzbibliotheken *pm2-windows-service* oder *pm2-windows-startup* vor. Letzteres nutzt keinen richtigen Windows Service, sondern hängt ein Start-Skript in die Registry. Für einen Produktiv-Betrieb mit Windows-Servern ist deshalb *pm2-windows-service* sinnvoller.

Hat man sämtliche gewünschten Anwendungen gestartet, kann man diesen Zustand mit dem Befehl *save* für Reboots sichern:

```
$ pm2 save

[PM2] Dumping processes
```

Je mehr Optionen man an PM2 übergibt, desto umständlicher wird die Konfiguration per Kommandozeile. Stattdessen kann man auch eine Deklaration im JSON-Format erzeugen. Dazu legt man eine Datei mit der Endung *.json* an und übergibt sie als Parameter an *pm2 start*:

```
$ pm2 start ecosystem.json
```

Die Bezeichnung *ecosystem* findet man häufig in der PM2-Dokumentation. Es ist eine gängige Konvention. Eine sehr kurze Konfiguration kann folgendermaßen aussehen:

```
{
  "name": "webapi",
  "script": "bin/api.js"
}
```

Mit *name* kann man der Anwendung einen Namen geben – genauso wie mit der CLI-Option *-n*. Das Anwendungsskript wird mit dem Attribut *script* konfiguriert.

Mit weiteren Attributen kann man das Startverzeichnis festlegen *(cwd)* und Umgebungsvariablen *(env)* oder Parameter *(args)* an das Anwendungsskripts übergeben. Auch für die Node.js-Engine kann man mit *node_args* Parameter setzen:

```
{
  "name": "webapi",
  "script": "bin/api.js",
  "cwd":"/opt/nodeapps/webapi",
  "env": {
    "NODE_ENV": "production",
    "API_TOKEN": "xyz"
  },
  "args":["--production"],
  "node_args":["--harmony"]
}
```

In dieser Konfiguration wird ein Microservice mit dem Namen *webapi* konfiguriert. Beim Starten wird Node.js mit dem Parameter *--harmony* und dem Skript *bin/api.js* aufgerufen. Die App befindet sich im Verzeichnis */opt/nodeapps/webapi*. Das Skript wertet Kommandozeilenparameter aus, wie das hier übergebene *--production*. Darüber hinaus finden auch Umgebungsvariablen wie *NODE_ENV* oder *API_TOKEN* Verwendung.

Apps neu starten bei Änderungen

PM2 kann Anwendungen automatisch neu starten, sobald sich ihre Quelltexte ändern. Das ist insbesondere dann sinnvoll, solange an Anwendungen aktiv entwickelt wird. Im Produktivbetrieb ergibt das nur bedingt Sinn, da sich die Quelldateien eigentlich nur beim Deployment neuer Versionen ändern würden.

Um automatisch bei Änderungen neu zu starten, schaltet man in den sogenannten *watch*-Modus. Eine Möglichkeit dazu ist der Kommandozeilenparameter *--watch*:

```
pm2 start bin/app.js --watch
```

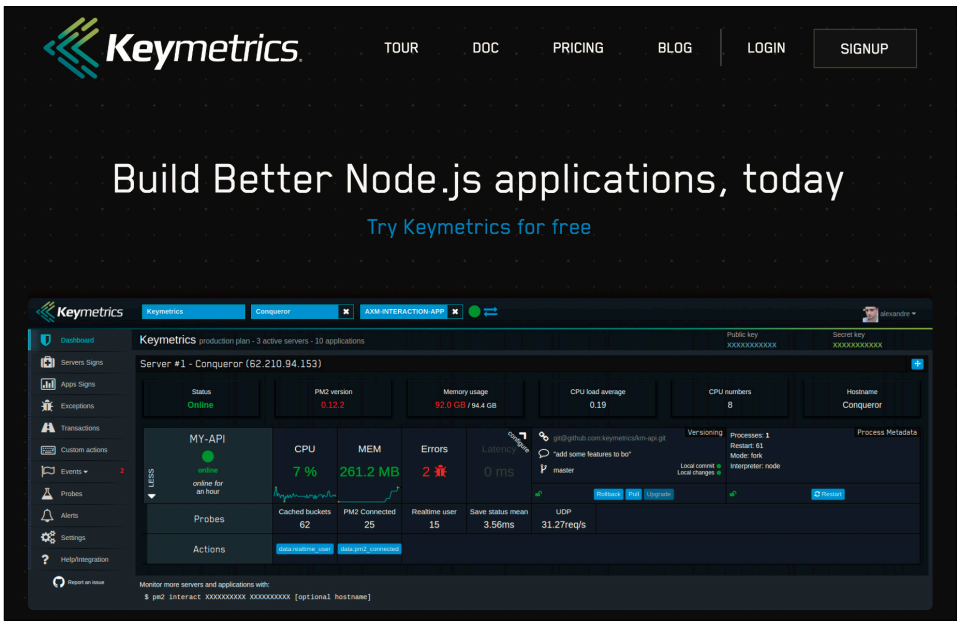
Diese Option sorgt dafür, dass die Anwendung neu geladen wird, sobald sich eine Datei im aktuellen Arbeitsverzeichnis oder irgendeinem darunter befindlichen Verzeichnis (rekursiv) ändert. Bei größeren Anwendungen ist eine präzisere Konfiguration des *watch*-Mode sinnvoll: Dazu muss man eine Anwendungskonfiguration festlegen:

```
{
  "name": "webapi",
  "script": "bin/api.js",
  "cwd":"/opt/nodeapps/webapi",
  "env": {
    "NODE_ENV": "production",
    "API_TOKEN": "xyz"
  },
  "args":["--production"],
  "node_args":["--harmony"]
  "watch": ["bin", "src"],
  "ignore_watch": ["node_modules"]
}
```

Mit der Option *watch* kann man ein Array von Verzeichnissen festlegen, das PM2 auf Änderungen beobachtet. Möchte man Verzeichnisse ignorieren, kann man sie per *ignore_watch* als Zeichenkette oder als Array aus Zeichenketten konfigurieren. Die einzelnen Zeichenketten können auch Muster oder reguläre Ausdrücke sein. ▶

Ein weiteres Neustartkriterium ist der Speicherverbrauch einer Anwendung. Gerade im Produktivbetrieb kann es sinnvoll sein, Anwendungen neu zu starten, sobald ihr Speicherverbrauch ein bestimmtes Limit erreicht hat. Dazu dient der Kommandozeilenparameter *--max_memory_restart* oder die Konfigurationsoption *max_memory_restart*:

```
{
  "name": "webapi",
  "script": "bin/api.js",
  "cwd":"/opt/nodeapps/
  webapi",
  ...
  "max\_memory\_restart":
  "64M"
}
```



Keymetrics ist ein kommerzieller Dashboard-Dienst für PM2 (Bild 2)

Mit dieser Konfiguration wird die Anwendung neu gestartet, sobald sie maximal 64 MByte Arbeitsspeicher belegt. PM2 prüft die Speicherbelegung etwa alle 30 Sekunden; das bedeutet, dass einerseits erst bis zu 30 Sekunden später bemerkt wird, dass das Limit erreicht wurde, und andererseits der Speicherverbrauch in diesem Zeitraum natürlich auch deutlich über die konfigurierte Grenze steigen kann. Fällt er bis zum Check gar wieder auf ein niedriges Niveau, dann wird die Anwendung auch nicht neu gestartet. Es handelt sich nicht um ein hartes Speicherverbrauchslimit, sondern um ein Feature, das für Anwendungen mit Speicherlecks gedacht ist. Auf diese Weise kann man beispielsweise die Produktivanwendung weiter betreiben, während an einem Fix für das Speicherleck gearbeitet wird.

Logdateien verwalten

Anwendungen, die im Hintergrund laufen, geben meist Meldungen in Logdateien aus. PM2 trägt dem Sorge, indem es die Programmausgaben automatisch in Logdateien umleitet und Kommandos anbietet, um diese Logs einzusehen:

```
$ pm2 logs
```

Dieser Befehl zeigt die Logausgaben aller Anwendungen gebündelt und in Echtzeit an. Dabei werden die Namen der Anwendungen vorangestellt und farbig ausgegeben (Ausgabe: Grün, Fehler: Rot). Mittels der Process-ID oder des zugewiesenen Namens kann man die Logausgaben auch auf einzelne Anwendungen begrenzen:

```
# Logs der Anwendung mit Prozess-ID 0
$ pm2 logs 0

# Log der Anwendung mit Name webapi
$ pm2 logs webapi
```

Um die Logs aller Anwendungen zu löschen, benutzt man das PM2-Kommando *flush*:

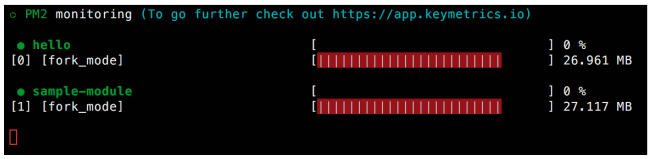
```
$ pm2 flush
```

Die Logdateien der einzelnen Anwendungen können in deren Konfigurationsdatei eingestellt werden:

```
{
  "script"          : "bin/webapi.js",
  "error_file"      : "webapi_errors.log",
  "out_file"        : "webapi.log",
  "log_date_format" : "YYYY-MM-DD HH:mm Z"
}
```

Die Eigenschaften *error_file* und *out_file* enthalten die Namen der jeweiligen Logdatei. Wählt man an dieser Stelle den gleichen Namen, dann loggt PM2 sowohl Fehler als auch einfache Ausgaben in eine Datei. Mit der Konfigurationsoption *log_date_format* kann man das ausgegebene Datumsformat beeinflussen.

Mit dem Kommandozeilenbefehl *monit* kann man die laufenden Anwendungen einer PM2-Instanz überwachen. Bild 1 zeigt, wie PM2 auf der Konsole den jeweiligen Speicherver-



Monit: Der PM2-Befehl *monit* visualisiert CPU-Last und Speicherverbrauch (Bild 1)

brauch und die CPU-Last der einzelnen Anwendungen visualisiert.

Wer mehr als diese einfache Konsolendarstellung möchte, der kann PM2 mit dem Monitoringdienst Keymetrics der PM2-Entwickler verbinden (Bild 2).

Dieses umfangreiche und erweiterbare Web-Dashboard ist allerdings nicht kostenlos. Man kann damit Anwendungen, die auf mehreren Servern verteilt sind, in einem gemeinsamen Dashboard komfortabel überwachen und steuern.

Webschnittstellen zu PM2

Als Alternativen gibt es eine Reihe von Open-Source-Projekten, die ebenfalls Webschnittstellen zu PM2 bereitstellen: Das Projekt pm2-web ist ein einfaches, webbasiertes Dashboard. Bild 3 zeigt die Oberfläche.

Die einzelnen Anwendungen werden aufgelistet und können gestoppt und gestartet werden. Klappt man die Zeile einer App heraus, dann zeigen Diagramme die CPU-Last und den Speicherverbrauch an. Die Installation des Dashboards ist sehr einfach:

```
$ npm install -g pm2-web
$ pm2-web
```

Die zweite Alternative ist pm2-gui. Wie pm2-web ist es ebenfalls leicht per NPM zu installieren. Bild 4 zeigt ein Curses-basiertes Terminal-Interface. Man kann jedoch auch eine Weboberfläche nutzen. Ähnlich wie pm2-web zeigt pm2-gui die Ressourcenbelegung des Servers an und listet die mit PM2 verwalteten Anwendungen auf (Bild 5). Auch mit pm2-gui kann man Anwendungen stoppen und starten und sich Speicher- sowie CPU-Diagramme anzeigen lassen (Bild 6).

Clustering-Modus

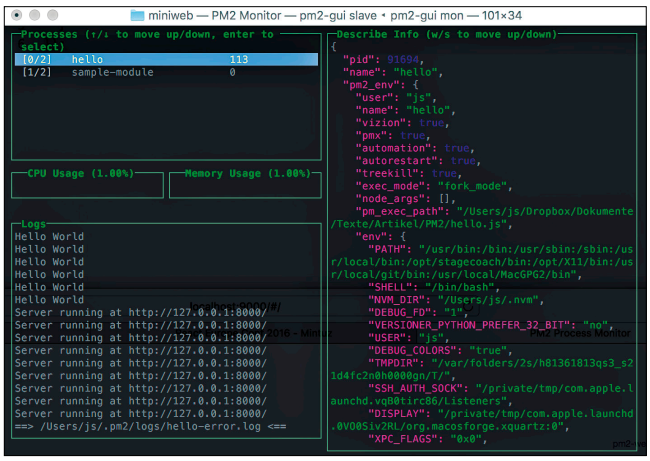
Eines der herausragenden Merkmale von PM2 ist der Clustering-Modus für Anwendungen. Unter der Voraussetzung, dass die Anwendung keinen eigenen Anwendungszustand besitzt, kann PM2 automatisch mehrere Instanzen starten und verwalten. Jede einzelne Instanz kann dann automatisch Anfragen verarbeiten.

Bild 7 zeigt einen Fall, bei dem Anwendungszustand auf einem der Knoten entsteht. Sobald ein Client bei einem Zugriff einen anderen Cluster-Knoten nutzt, ist der Anwendungszustand weg und die weitere Ausführung nicht möglich.

Bild 8 demonstriert, wie man den Anwendungszustand in eine Backend-Datenbank wie Redis oder MongoDB verlagern kann. Die einzelnen Node.js-Knoten des Clusters bleiben dann zustandslos. Gerade bei Node.js-Diensten ist der Clustering-Modus interessant: Anwendungen können auf diese Weise mehrere CPU-Kerne effizient nutzen.

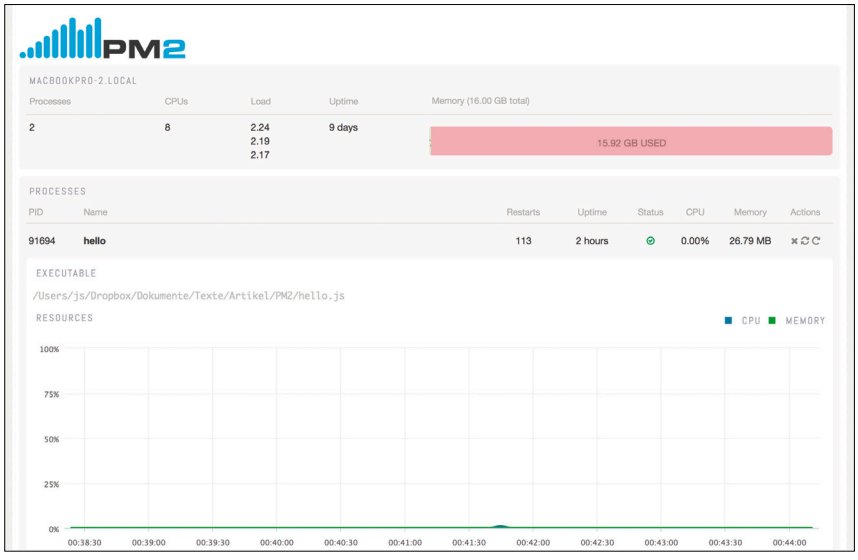
Um den Cluster-Modus für eine App einzustellen, bedient man sich der Konfigurationsoptionen *exec_mode* und *instances*:

```
{
  "script"       : "bin/webapi.js",
  "exec_mode"    : "cluster",
```



Das Terminal-Dashboard von pm2-gui (Bild 4)

Liste der Anwendungen im Webinterface von pm2-gui (Bild 5)



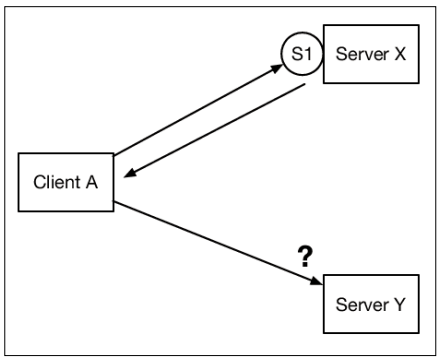
pm2-web: So präsentiert sich die Weboberfläche pm2-web (Bild 3)

```
  "instances"      : 4,
}
```

Wird diese Konfiguration gestartet, verwaltet PM2 vier Instanzen des WebAPI-Skripts gleichzeitig. Die Last verteilt sich auf diesen Prozessen.

Transpiling

Viele Compiler übersetzen nicht in Maschinensprache, sondern übersetzen von einer Hochsprache zu einer anderen Hochsprache. Häufig ist C das Compile-Ziel, da man so von den vielen Hardwareoptimierungen moderner C-Compiler profitieren kann.



Cluster mit Anwendungszustand pro Knoten (Bild 7)

Mittlerweile ist JavaScript, neben C, das gängigste Compile-Ziel. Für derartige Compiler hat sich mittlerweile der Begriff Transpiler herausgebildet. Neue Sprachen wie CoffeeScript, Elm, PureScript, ClojureScript oder TypeScript wurden von Grund auf so konzipiert, dass sie nach JavaScript übersetzt werden.

Auch altbekannte Vertreter wie OCaml können mittlerweile nach JavaScript übersetzt werden. Der Grund für diese Beliebtheit ist einfach: Moderne JavaScript-Engines sind hochoptimiert und noch dazu in Entwicklerkreisen extrem verbreitet. Desktop-PCs, Notebooks, Tablets, Fernseher, Uhren – es existiert nahezu kein Gerät, für das es nicht zumindest eine JavaScript-Engine gibt. Ob Webbrowser, Server oder native Desktop- und Mobile-Apps: Die Anwendungsfälle sind vielgestaltig.

Möchte man nun statt herkömmlichem JavaScript in PM2 mit einem solchen Transpiler arbeiten, so gibt es grundsätzlich zwei verschiedene Ansätze: Entweder man konfiguriert einen alternativen Interpreter oder man übersetzt die Quellsprache vorher in JavaScript, das der Node.js-Interpreter direkt versteht.

```
{
  "name": "webapi",
  "script": "bin/api.js",
  "cwd":"/opt/nodeapps/webapi",
  ...
  "exec_interpreter": "babel-node"
}
```

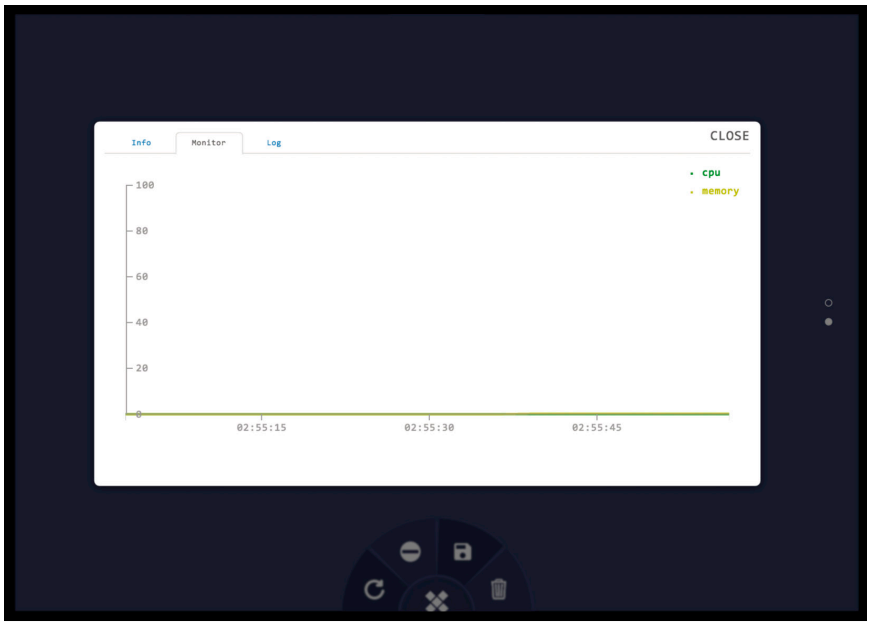
Diese Konfiguration tauscht mit der Option *exec_interpreter* den standardmäßig vorgelegten Interpreter *node* durch *babel-node* aus. Als Skript kann man dann einfach ES2015-Code nutzen, der dann durch Babel vor der Ausführung noch übersetzt wird. Konfiguriert man stattdessen den Interpreter *coffee* der Sprache CoffeeScript, dann kann man stattdessen Coffee-Skripts eintragen.

Streng genommen muss der Interpreter nicht einmal ein JavaScript-Transpiler sein. Man kann ebensogut *python* oder *ruby* eintragen und damit in diesen Sprachen geschriebene Anwendungen mit PM2 starten:

```
# file: helloweb.rb
require 'webrick'
include WEBrick
puts "Starting server:
http://#{Socket.gethostname}:3000"
server = HTTPServer.new(:Port=>3000,:
DocumentRoot=>Dir::pwd )
trap("INT"){ server.shutdown }
server.start
```

Dieser Ruby-Code startet eine Instanz des WEBrick HTTP-Servers und verwendet das aktuelle Verzeichnis als Dokument-Wurzelverzeichnis. Mit der folgenden PM2-Konfiguration kann der Dienst verwaltet werden:

```
{
  "name": "helloweb",
  "script": "helloweb.rb",
  ...
  "exec_interpreter": "ruby"
}
```



Charts mit CPU-Last und Speicherverbrauch (Bild 6)

Man sollte jedoch eines stets bedenken: Sobald man mit *exec_interpreter* einen anderen Interpreter als *node* wählt, funktioniert der Clustering-Modus nicht mehr. Diese Variante eignet sich deshalb insbesondere während der Entwicklung einer Anwendung.

Eine andere Möglichkeit bewahrt den Cluster-Modus: Einige Transpiler können sich in die *require*-Funktion von

Node.js einklinken. Die gepatchte *require*-Funktion sorgt dann dafür, dass geladene Module zuerst durch den Transpiler transformiert werden:

```
require("babel-register");
require("src/api")
// <-- Modul mit ES2015 Code
```

Sobald eine neue Anwendung oder eine neue Version einer bestehenden Anwendung einen stabilen Zustand erreicht, geht es um die Auslieferung. Die Anwendung muss beispielsweise auf einen oder mehrere Server geladen werden. Eine dort bereits laufende Version kann anschließend durch die neue Version ersetzt werden. PM2 bietet zur Verteilung von Anwendungen das Kommando *deploy*. Die Grundlage dafür ist eine erweiterte JSON-Anwendungskonfiguration:

```
{
  apps: [{
    "name": "webapi",
    "script": "bin/api.js",
    "cwd":"/opt/nodeapps/webapi",
    "env": {
      "NODE_ENV": "production",
      "API_TOKEN": "xyz"
    },
    "args":["--production"],
    "node_args":["--harmony"]
  }],
  "deploy": {
    "production": {
      "user": "nodeapps",
      "host": ["example.com"],
      "ref": "origin/master",
      "repo": "git@github.com/user/webapi.git",
      "path": "/opt/nodeapps/webapi",
      "post-deploy": "npm install && pm2
      startOrRestart ecosystem.json -env production"
    }
  }
}
```

Unter dem Key *apps* kann man eine oder mehrere Anwendungskonfigurationen angeben. Das ist keine Besonderheit des Deployments – auch beim Starten von Anwendungen per JSON-Konfiguration kann man so mehrere Anwendungen innerhalb einer Datei vorbereiten. Damit PM2 weiß, wo es die Anwendungen deployen soll, gibt man mit dem Schlüssel *deploy* eine oder mehrere Zielumgebungen an.

Im Beispiel gibt es die Umgebung *production*: Der Zielhost hat den Namen *example.com* und die Anwendung soll als Benutzer *nodeapps* unter */opt/nodeapps/webapi* installiert und gestartet werden. Die zugehörigen Quellen liegen unter dem Git-Repo *git@github.com/user/webapi.git*. PM2 klont das Repository und trackt dann den Branch *origin/master*.

Mit der Option *post-deploy* kann man – wie hier im Beispiel – noch die notwendigen NPM-Module installieren. PM2 führt die dort angegebenen Befehle nach dem Deployment aus.

Um auf dem Server ein initiales Ordner-Setup zu erzeugen, führt man folgenden Befehl aus:

```
$ pm2 deploy ecosystem.json production setup
```

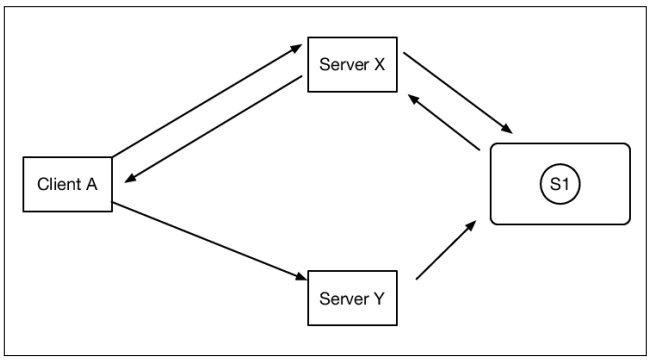
Danach kann man jederzeit eine neue Version ausliefern.

nginx als Frontend-Webserver

Möchte man mit PM2 auf einem Server mehrere Webservices betreiben, dann bietet es sich an, einen Frontend-Webserver wie nginx als Load-Balancer vorzuschalten. Damit hat man auch gleichzeitig eine elegante Lösung, um den Port 80 zugreifbar zu machen, denn dieser ist üblicherweise ja privilegiert. Die nginx-Konfiguration in Listing 1 zeigt, wie man nginx so einstellt, dass es Anfragen an einen sogenannten Upstream weiterleitet. ▶

Listing 1: nginx-Konfiguration

```
upstream my_nodejs_upstream {
  server 127.0.0.1:3001;
}
server {
  listen 80;
  server_name ;
  root /opt/nodeapps/webapi/www;
  location / {
    proxy_set_header X-Forwarded-For
    $proxy_add_x_forwarded_for;
    proxy_set_header Host $http_host;
    proxy_set_header X-NginX-Proxy true;
    proxy_http_version 1.1;
    proxy_set_header Upgrade $http_upgrade;
    proxy_set_header Connection "upgrade";
    proxy_max_temp_file_size 0;
    proxy_pass http://my_nodejs_upstream/;
    proxy_redirect off;
    proxy_read_timeout 240s;
  }
}
```



Cluster ohne Anwendungszustand pro Knoten (Bild 8)

Der mit PM2 gestartete Webservice wird dann auf localhost gebunden und über den Port 3001 zur Verfügung gestellt. Wichtig ist dabei insbesondere auch der Header *X-Forwarded-For:* Er sorgt dafür, das der Upstream die IP-Adresse, auf der die Anfrage ankam, erhält – sonst würde stattdessen die IP-Adresse des nginx angenommen werden.

Prozessmanager per API nutzen

PM2 ist nicht nur in Node.js implementiert, es stellt auch ein eigenes API bereit. Damit kann man den Prozessmanager in eigenen Node.js-basierten Projekten nutzen.

Möchte man das API in einem Node-Modul nutzen, so trägt man das Paket *pm* als Abhängigkeit in der *package.json* ein. Dies geschieht automatisch wenn man PM2 im Projektverzeichnis mit dem Attribut *--save* installiert:

```
$ npm install pm2 --save
```

Im Code lädt man das API-Objekt wie gewohnt mit der *require()*-Funktion. Die Methode *connect* stellt eine Verbindung zu einem PM2-Daemon her und startet diesen bei Bedarf vorher. Anwendungen können mit *start* gestartet und mit *stop* gestoppt werden (Listing 2).

Das Listing für *pm2api.js* stellt zuerst mit *connect* eine Verbindung zum PM2-Daemon her. Dann startet es eine App, deren Skript in der Datei *hello.js* liegt. Nachdem dieses Skript erfolgreich gestartet wurde, wird mit der Methode *describe* ein Informationsobjekt über den Prozess abgerufen. Dieses Objekt wird als JSON-Literal auf die Konsole ausgegeben:

```
$ node --harmony_destructuring pm2api.js
[{
  "pid":89638,
  "name":"hello",
  "pm2_env": ...
]

Process finished with exit code 0
```

Listing 2: pm2api.js

```
"use strict";
const {connect, start, describe, disconnect}
  = require("pm2");

connect((err)=> {
  if (err) {
    console.error(err);
    process.exit(2);
  }
  start({
    script: process.cwd() + "/hello.js"
  }, (err, [proc])=> {
    if (err) throw err;
    describe(proc.name, (err, info)=> {
      console.log(JSON.stringify(info));
      disconnect();
    });

  });
});
```

Listing 3: Modul-Vorlage generieren

```
cd /opt/nodeapps
$ pm2 module:generate miniweb
└─┬ pmx@0.5.0
  ├─┬ debug@2.2.0
  │ └── ms@0.7.1
  └── json-stringify-safe@5.0.1

[PM2][Module] Module sample created in folder:  /
opt/nodeapps/miniweb

Start module in development mode:
$ cd miniweb/
$ pm2 install .

Module Log:
$ pm2 logs miniweb

Uninstall module:
$ pm2 uninstall miniweb

Force restart:
$ pm2 restart miniweb
```

Die Verbindung zum Daemon sollte stets wieder mit *disconnect* beendet werden.

PM2 durch Module erweitern

PM2 kann durch Module erweitert werden. Dabei sind PM2-Module nichts anderes als gewöhnliche, über NPM verteilte Node.js-Pakete mit der Abhängigkeit zum Paket *pmx* und einer PM2-spezifischen Konfigurationssektion in der *package.json*.

PM2-Module können mit dem PM2-Befehl *install* installiert und mit *uninstall* wieder deinstalliert werden:

```
$ pm2 install <some-module>
$ pm2 uninstall <some-module>
```

Der Unterschied zu einer Installation per NPM? Das Modul wird bereits bei der Installation bei PM2 registriert und bei der Deinstallation auch wieder entfernt. Entwickler können mit dem Befehl *module:generate* eine Modul-Vorlage generieren lassen (Listing 3).

Der Generator erzeugt ein Verzeichnis *miniweb* und legt darin die Dateien *package.json* und *app.js* an. Listing 4 zeigt die generierte *package.json*. ▶

Dabei fällt auf, dass PM2 den Namen nicht anhand des eigentlich übergebenen Modulnamens *(miniweb)* gesetzt hat; diese Änderungen werden damit dem Entwickler überlassen.

Neben der vorbelegten Abhängigkeit *pmx* gibt es noch die Sektionen *config* und *apps*. Letzteres entspricht der bekannten JSON-Konfiguration. Dort wird also exakt festgelegt, wie die durch dieses Modul bereitgestellten Anwendungen gestartet und verwaltet werden sollen. Im generierten Beispiel wird das Skript *app.js* gestartet, und falls es 200 MByte erreicht, wird es neu gestartet.

Das generierte Modul kann im Modulverzeichnis mit folgendem Befehl installiert werden:

```
$ pm2 install
```

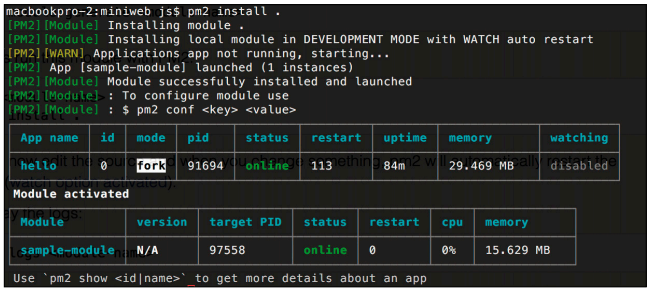
Bild 9 zeigt, wie PM2 das Modul im Entwicklermodus installiert, den *WATCH*-Modus aktiviert und die Anwendung startet. Bei Änderungen im Sourcecode wird die App automatisch neu gestartet.

Die pm2-hive-Website auf GitHub (https://github.com/pm2-hive) listet eine Reihe von offiziellen PM2-Modulen. Der Großteil davon sind letztlich Erweiterungen für das Keymetrics-Dashboard – was sicherlich auch als Hauptzweck dieses Modulsystems gelten kann. Dennoch ist dies nicht zwingend notwendig.

Ein sehr interessantes, von Keymetrics unabhängiges Modul ist beispielsweise pm2-intercom. Damit können die mit PM2 verwalteten Prozesse untereinander Nachrichten aus-

Listing 4: package.json

```
{
  "name": "sample-module",
  "version": "1.0.0",
  "description": "PM2 Sample Module",
  "main": "app.js",
  "dependencies": {
    "pmx" : "beta"
  },
  "repository": {
    "type": "git",
    "url": "https://github.com/keymetrics/pmx.git"
  },
  "config" : {
    "conf_var_1" : true,
    "conf_var_2" : "myvalue"
  },
  "apps" : [{
    "merge_logs"         : true,
    "max_memory_restart" : "200M",
    "script"             : "app.js"
  }],
  "author": "Keymetrics Inc.",
  "license": "MIT"
}
```



Installation eines lokalen PM2-Moduls (Bild 9)

tauschen. Ein Prozess registriert sich dabei für bestimmte Themen:

```
process.on('message', function(packet) {
  if (packet.topic == 'cmd:new-data-available') {
    console.log(,New Data: '' packet.data);
  }
});
```

Mit *process.on* lauscht der aktuelle Prozess auf Ereignisse vom Typ *message*. Sobald er ein solches trifft, begutachtet er das mitgelieferte Paket und handelt entsprechend. Das Senden einer zu diesem Beispiel passenden Nachricht geht so:

```
process.send({
  topic : 'cmd:new-data-available', data : {
    payload : '12345' }
});
```

PM2 ist ein flexibler, stabiler Prozessmanager für Node.js-Anwendungen. Die Installation ist einfach, und sowohl das API als auch eine große Zahl von Erweiterungen erlauben gute Integrationsmöglichkeiten in bestehende Infrastrukturen. Der Monitoringdienst Keymetrics ist komfortabel. Wer dennoch lieber auf eine kostenlose Lösung abzielt, der kann sich unter anderem pm2-web oder pm2-gui näher ansehen.

Ob man für die Verwaltung von Node.js-basierten Serverdiensten wirklich einen Prozessmanager wie PM2 benötigt, ist jedoch auch mitunter fraglich. Seit einiger Zeit finden Container-Lösungen wie Docker immer mehr Verbreitung. In einer solchen Containerumgebung ist ein Prozessmanager nicht sinnvoll. Allerdings ist die Lernkurve bei Docker durchaus auch abschreckend und ein PM2 für viele Anwendungsfälle immer noch die einfachere Lösung. ■

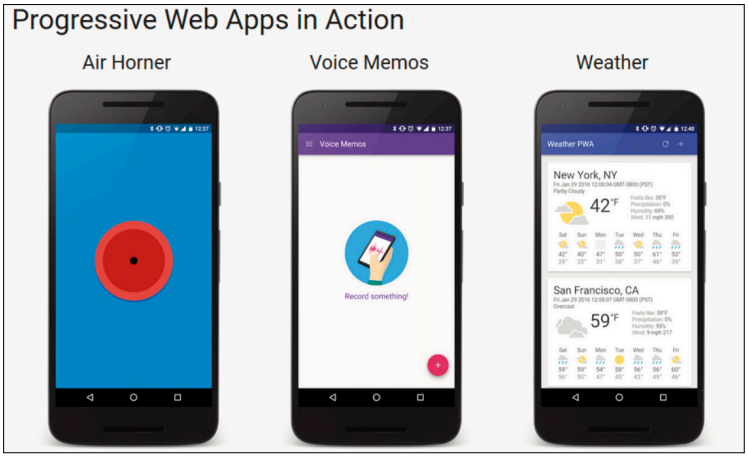
Jochen H. Schmidt
ist als Autor, Berater und Software-Entwickler tätig. Schwerpunkte seiner Aktivitäten sind Webentwicklung und Webtechnologien. Er ist Verfasser von bekannten Fachbüchern zum Thema Common Lisp.

OFFLINE-FÄHIGE WEB-APPS

Progressive Web Apps

Ein neuer Ansatz für Web-Apps kommt mit Progressive Web Apps von Google.

Native Apps sind cool, aber in mehrerlei Hinsicht kostenintensiv. Ein Problem an nativen Apps ist beispielsweise, dass es nicht einfach ist, Nutzer zu gewinnen, die die App regelmäßig einsetzen. Fiksu, das Statistiken im Zusammenhang mit Apps anbietet, ermittelt, was es kostet, einen regelmäßigen Nutzer für eine App zu gewinnen. Der Preis bewegt sich bei iOS-Nutzern zwischen 2,50 und 4,50 Dollar. Problematisch ist auch, dass nur ein geringer Teil der Leute, die die Seite der App im App Store besuchen, diese dann auch wirklich nutzen. Denn dazu sind mehrere Einzelschritte nötig: Nachdem man die App gefunden hat, muss man sie herunterladen, installieren, die Berechtigungen akzeptieren et cetera – und jeder Schritt kostet User. Am Schluss bleiben nur 20 Prozent oder sogar noch weniger übrig (Bild 1).



Beispiele für Progressive Web Apps (Bild 2)

Ein Versuch, diese Verluste zu reduzieren, ist die Fullscreen-Werbung für die native App, die Smartphone-Nutzern auf manchen Seiten präsentiert wird. Aber das ist prinzipiell keine gute Idee: Zum einen verärgert es die Besucher, und zum anderen straft Google solche Seiten seit September 2015 ab, indem sie nicht mehr als mobilfreundlich eingestuft werden.

Es gab und gibt immer wieder Versuche, Web-Apps zu etablieren. Native Apps müssen für die verschiedenen Plattformen einzeln programmiert werden, weil unterschiedliche Techniken benötigt werden. Im Gegensatz dazu basieren Web-Apps auf den klassischen Webtechnologien: HTML, CSS, JavaScript. Sie haben aber – im Gegensatz zu Webseiten – zentrale App-Eigenschaften.

Die erste und wichtigste Anforderung: Die Web-App muss auch offline funktionieren. Oft betrachtet man online und offline als Gegensätze, dabei gibt es natürlich auch die Zwischenzustände, das heißt, eine schlechte Verbindung beispielsweise auf dem Land. Entscheidend ist, dass die Web-Apps auch dann funktionieren.

Ein weiteres unverzichtbares Merkmal ist, dass Apps auf dem Startbildschirm erscheinen und vom Benutzer darüber aufgerufen werden können. Nach dem Aufruf soll sich die Web-App zudem wie eine App verhalten, das heißt, die Browseroberfläche soll nicht unbedingt sichtbar bleiben.

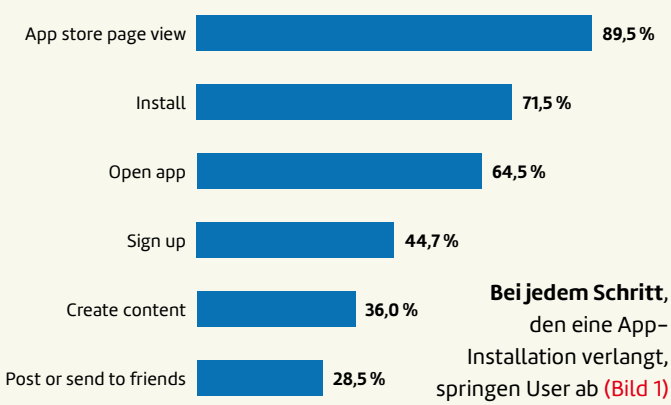
Ein neuer Ansatz für Web-Apps stammt von Google und nennt sich Progressive Web Apps.

Progressive Web Apps

Progressive Web Apps (Bild 2) zeichnen sich durch eine Reihe spezieller Eigenschaften aus:

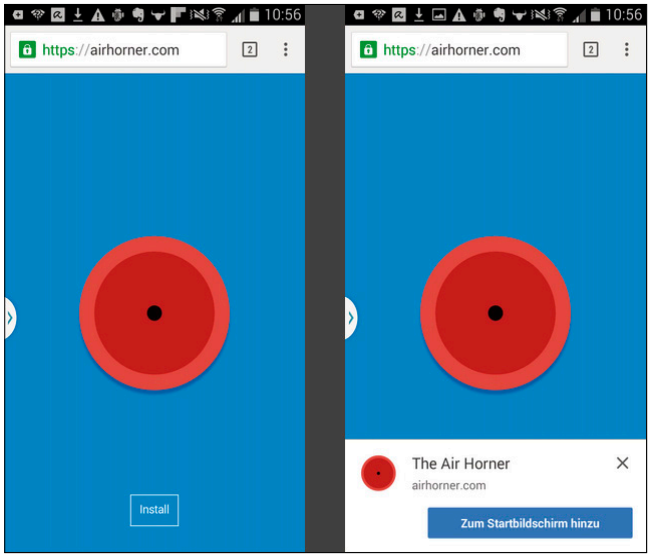
- Der Namensbestandteil »progressive« verweist darauf, dass Progressive Web Apps dem Prinzip des Progressive Enhancement folgen: Sie verwenden fortgeschrittene Techniken, die jedoch keine Voraussetzung für die Basisfunktionalität sind. Damit laufen Web-Apps prinzipiell überall, aber von den fortgeschrittenen Features profitieren nur die fähigeren Browser.
- Progressive Web Apps setzen auf Responsive Webdesign, sodass sie auf den unterschiedlichsten Geräten angezeigt werden können.
- Sie funktionieren unabhängig von der vorhandenen Verbindung – also auch offline und bei schlechter Verbindung.

User-Verhalten bei der App-Installation



Bei jedem Schritt, den eine App-Installation verlangt, springen User ab (Bild 1)

web & mobile developer 8/2016



Aufforderung, die Seite zum Startbildschirm hinzuzufügen (Bild 3)

- Dank der JavaScript-API Service Worker bieten Progressive Web Apps immer die aktuellen Inhalte.
- Da die Progressive Web Apps nur über HTTPS ausgeliefert werden, sind sie sicher.
- Der Benutzer kann diese Progressive Web Apps direkt installieren und zum Startbildschirm hinzufügen – ohne Umweg über App Stores.
- Progressive Web Apps sind für Suchmaschinen auffindbar – auch dank des sogenannten Web Manifests, das sie beschreibt.
- Progressive Web Apps sind wie normale Webseiten über Links erreichbar. Diese Links können verteilt und somit die Web-Apps einfach weiterempfohlen werden.

Ein wichtiges Feature von Progressive Web Apps ist, dass sie sich sehr komfortabel installieren und zum Startbildschirm hinzufügen lassen. Wenn der Benutzer mehrmals die Seite besucht hat, erhält er einen Install-Hinweis, und nach Klick darauf erscheint ein Banner, das das Hinzufügen zum Startbildschirm anbietet (Bild 3).

Installationsbanner

Wenn der Benutzer die Web-App danach über das Icon auf dem Startbildschirm ausführt, erscheint ein Splashscreen, bis das Programm geladen ist. Die Web-App kann im Vollbildschirm-Modus ausgeführt werden und die Ausrichtung (Portrait/Landscape) lässt sich steuern.

Für die Anzeige des Installationsbanners gibt es mehrere Bedingungen:

- Die Webseite muss einen Service Worker einsetzen.
- Das Installationsbanner wird nur auf sicheren, per HTTPS ausgelieferten Seiten angezeigt.
- Zudem ist eine JSON-Datei mit Informationen über die Web-App notwendig.

Das Banner wird außerdem nur eingeblendet, wenn ein mutmaßliches Interesse des Benutzers besteht. Bei Opera etwa wird das dann angenommen, wenn der Besucher die Webseite wenigstens zweimal besucht hat und wenn zwischen den einzelnen Besuchen mindestens fünf Minuten liegen. Dieselben Bedingungen gelten auch für Chrome – allerdings findet sich immer der Hinweis, dass die Bedingungen sich ändern können.

Kommen wir nun zur JSON-Datei mit Informationen über die Web-App, das sogenannte Web Manifest. Listing 1 zeigt ein Beispiel.

Sehen wir uns die einzelnen Bestandteile des Manifests an: *name* ist der vollständige Name, im Gegensatz zum kürzeren *short_name*. Der kürzere Name wird auf dem Startbildschirm angezeigt.

Außerdem benötigen Sie Icons: Im Beispiel werden eine ganze Reihe von Icons für die verschiedenen Auflösungen der Display-Anzeigen spezifiziert, wobei die Größe von 192 x 192 die wichtigste ist.

Meist soll beim Start der App eine bestimmte Seite zuerst aufgerufen werden. Diese können Sie über *start_url* bestimmen. Außerdem können Sie dadurch das Manifest auf allen Seiten integrieren und trotzdem sicherstellen, dass beim Öffnen der Web-App die richtige Startseite erscheint und nicht diejenige Seite, über die die Web-App installiert wurde. ▶

Listing 1: Manifest

```
{
  "name": "WebApp Beispiel",
  "short_name": "Beispiel",
  "icons": [{
    "src": "images/icons/icon-128x128.png",
      "sizes": "128x128",
      "type": "image/png"
    }, {
      "src": "images/icons/icon-144x144.png",
      "sizes": "144x144",
      "type": "image/png"
    }, {
      "src": "images/icons/icon-152x152.png",
      "sizes": "152x152",
      "type": "image/png"
    }, {
      "src": "images/touch/icon-192x192.png",
      "sizes": "192x192",
      "type": "image/png"
    }, {
      "src": "images/touch/icon-256x256.png",
      "sizes": "256x256",
      "type": "image/png"
    }],
  "start_url": "/index.html",
  "display": "standalone",
  "background_color": "#3E4EB8",
  "theme_color": "#2F3BA2"
}
```

Bei *display* gibt es drei mögliche Werte:

- **standalone** versteckt das Browser-UI und zeigt nur die Leiste oben und die Navigationsleiste unten.
- **fullscreen** – der Vollbildmodus wäre beispielsweise bei einem Spiel sinnvoll.
- **browser** zeigt die Web-App im normalen Browserfenster an. Das sieht zwar weniger App-typisch aus, hat jedoch den Vorteil, dass Sie keine eigenen Navigationselemente erstellen müssen, sondern die des Browsers verwenden können.

Sehr praktisch ist es, dass Sie mit CSS den Darstellungsmodus abfragen können. Durch folgenden Code lassen sich besondere Formatierungen definieren, wenn die Anwendung im Standalone-Modus angezeigt wird.

```
@media (display-mode: standalone) {
  /* Formatierungen für Standalone-Modus */
}
```

Entsprechend sind auch die anderen Display-Modi wie *browser* oder *fullscreen* abfragbar. Die Farbangaben *(background_color* und *theme_color)* sind für den Splashscreen wichtig. Für den Splashscreen wird der vollständige Name der Applikation benutzt, ein Icon aus dem Icon-Array und eine Hintergrundfarbe.

Schließlich müssen Sie den Browser noch auf das Manifest hinweisen. Das erledigt ein *link*-Element mit *rel="manifest"*:

```
<link rel="manifest" href="/manifest.json">
```

Dies sollte auf allen Unterseiten stehen und auf die Manifest-Datei im JSON-Format verweisen, die üblicherweise im Root-Verzeichnis abgespeichert ist.

Hilfe beim Manifest

Damit bei der Erstellung des Manifests nichts schiefgeht, können Sie den Manifest-Generator von Bruce Lawson einsetzen und den Manifest-Validator von Mounir Lamouri zur Überprüfung nutzen.

Bei Tests ist es mühsam sicherzustellen, dass die nichttechnische Bedingung erfüllt ist, das heißt, die Anzahl an Besuchen, die die Browserhersteller dafür als notwendig erachten. Bei der Entwicklung lässt sich diese Überprüfung deaktivieren. Wählen Sie in Chrome:

```
chrome://flags/
```

Dann können Sie nach der Option Nutzerinteraktionsprüfungen umgehen *(#bypass-app-banner-engagement-checks)* suchen. Wenn Sie die Umgehung aktivieren, wird das Banner direkt beim ersten Besuch angezeigt. Hilfreich für Tests ist außerdem, die Bannerfunktionalität auch auf dem Desktop zu aktivieren. Das können Sie ebenfalls über *chrome://flags/* bewirken, indem Sie *Zu Ablage hinzufügen (#enable-add-to-shelf)* auf aktiviert schalten. Damit die Änderungen wirksam sind, muss Chrome erneut gestartet werden.

Wenn eine Webseite die Bedingungen erfüllt und ein Manifest vorhanden ist, erscheint im Chrome auf dem Desktop die etwas seltsam klingende Meldung: *Fügen Sie diese Website Ihrer Ablage hinzu, um sie jederzeit zu verwenden*. Wenn Sie das akzeptieren, wird eine Verknüpfung auf dem Desktop angelegt und Sie wissen, dass Ihr Manifest funktioniert.

Alternativen zum JSON-Manifest

Für Browser, die das Manifest für das Installationsbanner noch nicht unterstützen, kann die App natürlich auf die klassische Art über das Menü zum Startbildschirm hinzugefügt werden. Für iOS-Safari schreiben Sie dafür beispielsweise:

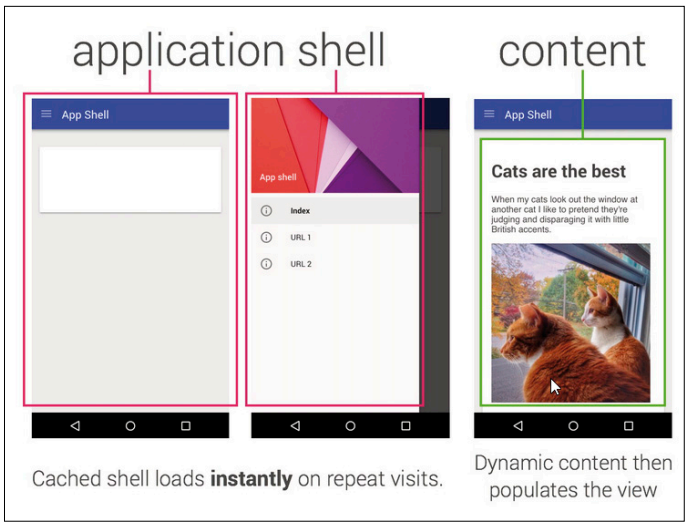
```
<meta name="apple-mobile-web-app-capable" content="yes">
<meta name="apple-mobile-web-app-status-bar-style"
content="black">
<meta name="apple-mobile-web-app-title"
content="Example App">
<link rel="apple-touch-icon" href="images/icons/
icon-152x152.png">
```

Und für Windows sind folgende Angaben gedacht:

```
<meta name="msapplication-TileImage" content="images/
icons/icon-144x144.png">
<meta name="msapplication-TileColor" content="#2F3BA2">
```

Web-Apps sollen schnell geladen werden. Dafür muss man zwischen den Bestandteilen unterscheiden, die aktualisiert werden müssen, und denen, die immer gleich bleiben. Die AppShell-Architektur hilft bei dieser Differenzierung.

Als Application Shell, kurz AppShell, wird das Gerüst der Anwendung bezeichnet, das heißt der minimale HTML-, CSS- und JavaScript-Code, der für die Oberfläche der Web-App benötigt wird. Dieser wird unterschieden vom eigentlichen Inhalt, der beispielsweise auch aktualisiert werden muss (Bild 4). Die AppShell sollte beim ersten Mal schnell laden und dann gecacht werden. Das sorgt für eine gute Per-



Application Shell und eigentlicher Inhalt (Bild 4)

formance, da die gecachten Elemente bei einem erneuten Besuch direkt angezeigt werden können, ohne dass auf externe Daten gewartet werden muss.

Zur AppShell gehört üblicherweise der Rahmen mit Navigationselementen, der auch bei verschiedenen Unterseiten vorhanden ist. Der AppShell-Code entspricht damit dem Code, den man in einem App Store veröffentlicht, wenn man eine native App erstellt.

Die Unterscheidung zwischen dynamischen Inhalten und AppShell ist natürlich nur bei dynamischen Seiten sinnvoll. Bei einer kleinen, statischen Seite könnte hingegen die gesamte Seite gecacht werden.

Offline: klassisch

Eine zentrale Anforderung an Web-Apps ist, dass sie offlinefähig sind. Um zu steuern, welche Dateien offline zur Verfügung stehen sollen, hat man ursprünglich auf AppCache gesetzt. Bei AppCache geben Sie in einem Cache-Manifest die Dateien an, die offline benötigt werden, sowie diejenigen, die über das Netzwerk geladen werden sollen, und zudem die Fallbacklösungen. Das kann beispielsweise folgendermaßen aussehen:

```
CACHE MANIFEST
# 2016-06-18:v2
CACHE:
index.html
stylesheet.css
scripts/main.js
NETWORK:
login.php
FALLBACK:
/ /static.html
```

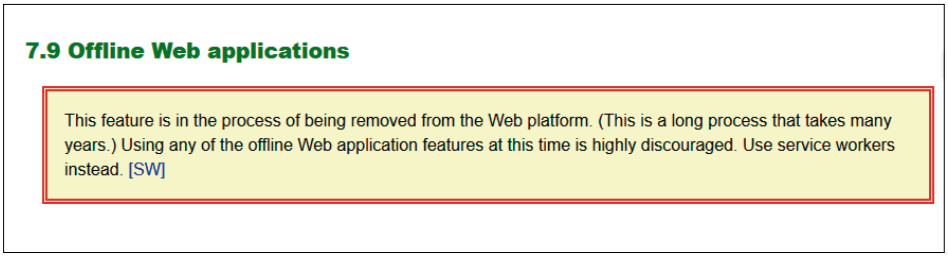
Auf diese Datei wird im HTML-Dokument verlinkt:

```
<html manifest="beispiel.appcache">
```

Wunderbar einfach in der Theorie, birgt AppCache jedoch in der Praxis so manche böse Überraschung, weil sich vieles nicht so genau steuern lässt, wie man es brauchen würde. Deswegen hat man sich inzwischen von AppCache verabschiedet. In der HTML-Spezifikation heißt es dazu unmissverständlich (Bild 5): »Dieses Feature wird aus der Web Plattform entfernt. […] Nutzen Sie stattdessen Service Workers.« Zur Beruhigung: Der Prozess des Entfernens kann auch mehrere Jahre dauern. Es empfiehlt sich also, für die Steuerung des Offline-Verhaltens Service Worker zu verwenden.

Service Worker API

Service Workers sind ein neues JavaScript-API, das sich wie ein Proxyserver zwischen Ihrer Webseite und dem Netzwerk befindet und mit dessen Hilfe Sie Ressourcen anfordern und variabel reagieren können. Zudem ermöglichen Service Workers einen Zugriff auf Push-Notifikationen und das Background sync-API.



Die WHATWG-Spezifikation ist eindeutig: AppCache wird entfernt (Bild 5)

Service Workers können das, was AppCache kann, aber auch noch wesentlich mehr. Und entscheidend ist dabei: Es gibt eine Reihe von APIs, die zu Service Workers gehören oder in irgendeiner Art damit verbunden sind. Besonders wichtig sind *cache* (Speichermechanismus) und *fetch* (Anfordern von Ressourcen).

Ein Service Worker läuft in einem eigenen Thread, unabhängig von der Seite, die ihn kontrolliert. Ein Service Worker hat damit keinen DOM-Zugriff und es sollte kein blockendes JavaScript genutzt werden, das heißt, Sie sollten keine synchronen APIs einsetzen. Beim Speichern von Daten sollte man deshalb auf *localStorage* verzichten und sich besser für IndexedDB entscheiden. Eine weitere Besonderheit: Service Workers funktionieren nur bei HTTPS; für Tests praktisch ist jedoch, dass es auch mit http://localhost klappt.

Sehen wir uns beispielhaft an, was man mit einem Service Worker machen kann. Eine vollständige Umsetzung einer Progressive Web App mit einem Service Worker finden Sie bei http://developers.google.com.

Ein Service Worker ist eine JavaScript-Datei. Um ihn zu aktivieren, muss er erst einmal registriert werden – was logischerweise in einer Datei außerhalb des Service Workers erfolgt.

Vor der Registrierung sollte überprüft werden, ob das API unterstützt wird – damit es keine Probleme in nicht unterstützenden Browsern gibt. Damit sieht die Registrierung so aus:

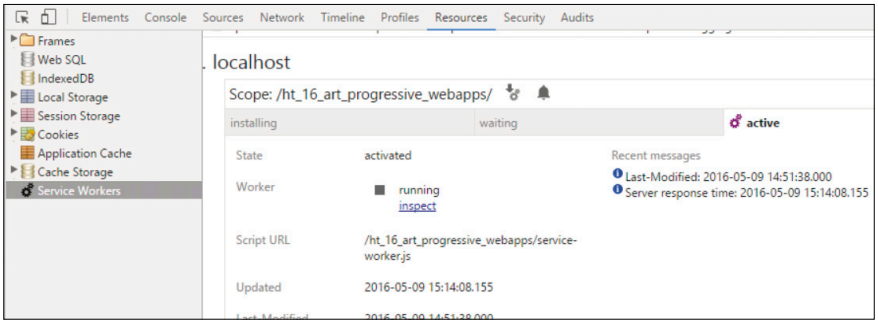
```
if('serviceWorker' in navigator) {
  navigator.serviceWorker
  .register('/service-worker.js')
  .then(function() { console.log('Service Worker
  Registered'); });
}
```

Im Beispiel befindet sich die *service-worker.js*-Datei im Root-Verzeichnis. Nach erfolgreicher Registrierung wird eine Meldung auf der Konsole ausgegeben.

Nachdem der Service Worker registriert ist und heruntergeladen wurde, wird ein *install*-Ereignis ausgelöst. Bei einer Web-App wäre es sinnvoll, auf dieses Ereignis zu reagieren, um die benötigten Dateien zu cachen. Im Unterschied zur Registrierung des Service Workers, die außerhalb der *service-worker.js*-Datei stattfindet, wird unsere Reaktion auf das *install*-Ereignis in die *service-worker.js*-Datei geschrieben. Zuerst werden zwei Variablen definiert: ein Name für ▶

den Cache und in *filesToCache* eine Liste der zu speichernden Dateien:

```
var cacheName = 'example';
var filesToCache = [
  '/index.html',
  '/styles/main.css',
  '/scripts/main.min.js'
];
```



In den Entwickler-Tools von Chrome finden sich unter *Resources* auch die Service Workers (Bild 6)

Dann können wir auf das install-Ereignis reagieren, eine Meldung ausgeben und die Dateien im Cache abspeichern. Das Ganze funktioniert allerdings nur, wenn wirklich alle Dateien gecacht werden können.

```
self.addEventListener('install', function(e) {
  console.log('[ServiceWorker] Install');
  e.waitUntil(caches.open(cacheName).
  then(function(cache) {
    console.log('[ServiceWorker] Caching app shell');
    return cache.addAll(filesToCache); }));
});
```

caches.open wird mit dem Cache-Namen aufgerufen, danach werden alle Dateien, die gecacht werden sollen, an *cache.addAll()* übergeben. Dabei wird auf die sogenannten Promises gesetzt. Promises repräsentieren das mögliche Ergebnis einer asynchronen Operation. Die *then()*-Methode gibt ein Promise-Objekt zurück.

Nach der Installation wird der Service Worker aktiviert (*activate*-Ereignis). Das *activate*-Ereignis könnte man zur Überprüfung einsetzen, ob die Dateien auf dem aktuellem Stand sind.

Fetch-Ereignisse

Mit Hilfe des Fetch-API können wir auf Anfragen reagieren, indem wir uns dann beispielsweise entscheiden, die zugehörigen Dateien aus dem Cache zu laden.

```
self.addEventListener('fetch', function(e) {
  console.log('[ServiceWorker] Fetch', e.request.url);
  e.respondWith(caches.match(e.request).
    then(function(response) {
      return response || fetch(e.request);
    })
  );
});
```

Hier wird überprüft, ob die angefragte Datei im Cache vorhanden ist. Wenn das der Fall ist, wird diese ausgeliefert, andernfalls wird die Datei normal aus dem Netzwerk angefordert.

Im Beispiel ist die gecachte Version immer die bevorzugte. Sie können aber auch festlegen, dass bestimmte Anforderungen immer zuerst an das Netzwerk gehen. Zu diesem Zweck können Sie bei Fetch über das Event-Objekt den *request.url* auslesen.

```
self.addEventListener('fetch', function(e) {
  var dataUrl = 'https://example.com/';
  if (e.request.url.indexOf(dataUrl) === 0) {

    // Hier Code für die aktuellen Daten

  } else {
    e.respondWith(caches.match(e.request).
      then(function(response) {
        return response || fetch(e.request);
      })
```

Links zum Thema

- Kosten für einen loyalen App-Nutzer
  https://www.fiksu.com/resources/fiksu-indexes
- Bei jedem Schritt verliert man Nutzer
  http://blog.gaborcselle.com/2012/10/every-step-costs-you-20-of-users.html
- Nicht mobilfreundlich: Webseiten mit ganzseitiger Werbung für Apps
  https://webmasters.googleblog.com/2015/09/mobile-friendly-web-pages-using-app.html
- Keine Zukunft für AppCache
  https://html.spec.whatwg.org/multipage/browsers.html#offline
- Eine Progressive Web App erstellen
  https://developers.google.com/web/fundamentals/getting-started/your-first-progressive-web-app
- Browserunterstützung einzelner Komponenten von Service Worker
  https://jakearchibald.github.io/isserviceworkerready
- Manifest-Generator
  https://brucelawson.github.io/manifest
- Manifest-Validator
  https://manifest-validator.appspot.com

Tabelle 1: Browserunterstützung der vorgestellten Techniken

Feature	Anmerkung	Chrome	Opera	Firefox	Safari	IE/Edge
Web Manifest	Informationen im JSON-Format über die Web-App; Voraussetzung für die Anzeige des Installationsbanners	ja	ja	–	–	–
navigator.serviceWorker	Grundlegende Unterstützung für das Service Worker API	ja	ja	ja	Under Consideration	Die Arbeit an der Umsetzung für Edge hat begonnen
Web Push API	Dieses API ermöglicht das Senden von Nachrichten vom Server an den Browser, selbst wenn die Seite nicht im Fokus oder nicht einmal geöffnet ist	Ja, aber *PushEvent.data* und *PushMessageData* werden nicht unterstützt		Ja, aber Browser muss gestartet sein, um Nachrichten zu empfangen	–	Under Consideration
Background sync	Erlaubt es, Aufgaben zu verschieben, bis wieder eine Verbindung besteht	ja	–	–	–	–

```
    );
  }
});
```

Bei der Arbeit mit Service Workers und der Fehlersuche hilft ein Aufruf von *about:serviceworkers* (Firefox), *browser://serviceworker-internals* (Opera) oder *chrome://serviceworker-internals/* (Chrome). Hier können Sie einzelne Service Workers abmelden.

Der Nachteil ist, dass Sie zum Debuggen ein neues Browserfenster öffnen müssen. Praktischer ist hingegen, dass die Service-Worker-Informationen sich inzwischen auch direkt über den Punkt *Resources* in den Chrome Entwickler-Tools anzeigen lassen. Sie sehen an dieser Stelle, ob die Installation und die Aktivierung des Service Worker geklappt haben (Bild 6) und können den Service Worker auch über den Button mit Pfeil und Zahnrad updaten.

Wie gesagt, gibt es eine Reihe von Techniken, die mit dem Service Worker API im Zusammenhang stehen. Sehr vielversprechend ist Background sync. Was damit gemeint ist, lässt sich am besten anhand des üblichen Mail-Ablaufs veranschaulichen: Um eine E-Mail zu versenden, muss eine Verbindung bestehen. Ist dies momentan nicht der Fall, wird die E-Mail üblicherweise in einem Postausgang gespeichert, von dem aus sie zu einem späteren Zeitpunkt gesendet werden kann, wenn wieder eine Verbindung besteht.

Ein solches Verhalten ist bei klassischen Webseiten schwer nachzubilden, da es nur funktioniert, solange – um beim Beispiel zu bleiben – der Postausgang im Browserfenster geöffnet ist. Dank Background sync soll so etwas im Hintergrund funktionieren, das heißt, man kann Aufgaben verschieben, bis eine Verbindung besteht.

Unbedingt zu erwähnen in diesem Zusammenhang ist das Push API: Sie ermöglicht es, dass die Web-App vom Server kommende Nachrichten empfängt, unabhängig davon, ob die Web-App gerade geöffnet ist oder ob überhaupt der Browser gestartet ist. Damit kann der Benutzer wichtige Updates erhalten, was wiederum die Useranbindung verbessert. Während *cache* und *fetch* in Chrome/Opera und Firefox implementiert sind, funktioniert die Push API nur teilweise in Firefox/Chrome und beispielsweise Background *sync* nur im Chrome.

Hilfreich in diesem Zusammenhang ist die Webseite https://jakearchibald.github.io/isserviceworkerready, die die Unterstützung einzelner Komponenten von Service Workers in den verschiedenen Browsern auflistet.

Fazit

Der neue Ansatz für Web-Apps erscheint vielversprechend. Die Aktivierung des Installationsbanners ist problemlos und der Benutzer kann die Web-App dadurch einfach installieren. Komplexer hingegen sind die Service Workers, dafür haben sie auch großes Potenzial. Überzeugend ist die Umsetzung der Offline-Fähigkeit. Gut, dass es eine Alternative zu dem doch im Handling ziemlich umständlichen AppCache gibt. Für AppCache spricht allerdings derzeit noch die breitere Browserunterstützung (Tabelle 1).

Dass die Auslieferung der Webseiten über HTTPS eine Voraussetzung für die Progressive Web Apps ist, leuchtet ein. Es zeigt einmal mehr, dass man an HTTPS nicht vorbeikommt, wenn man neue APIs einsetzen möchte. ■

Dr. Florence Maurice
ist Autorin, Trainerin und Programmiererin in München. Sie schreibt Bücher zu PHP und CSS3 und gibt Trainings per Video. Außerdem bloggt sie zu Webthemen unter:
http://maurice-web.de/blog

ES2015-PROMISE-PROGRAMMIERMUSTER

Vielversprechend

Seit ES2015 gehören Promises zum offiziellen Sprachstandard der Sprache ECMAScript.

Viele Entwickler tun sich jedoch schwer, das Programmierkonzept der Promises sinnvoll und richtig einzusetzen. Oft liest man Code, der Promises auf eine Art und Weise benutzt, wie sie nie gedacht waren. Unerwartetes Verhalten oder gar schwer auffindbare Fehler können die Konsequenz sein.

Als Webentwickler hat man sehr häufig mit asynchroner Programmierung zu tun. Einer der gängigsten Anwendungsfälle ist die Kommunikation mit einem Webserver. Sogenannte Single-Page-Anwendungen (SPA) rufen ihre Inhalte per Ajax vom Server ab. Auch auf dem Server kommt vorwiegend asynchrone Programmierung zum Einsatz: Denn Node.js arbeitet mit einer Ereignisschleife, weshalb es auf nicht blockierende, asynchrone APIs angewiesen ist.

Das wohl bekannteste Programmiermuster für asynchrone Programmierung in JavaScript sind die sogenannten Callbacks im Continuation Passing Style (CPS). Eine asynchrone Funktion nennt man so, wenn sie ihre Berechnung asynchron durchführt. Bei CPS nimmt sie dazu mittels eines zusätzlichen Parameters eine weitere Funktion entgegen: die Continuation oder Fortsetzung. Die Continuation-Funktion wird mit dem Ergebnis der Berechnung aufgerufen, sobald es zur Verfügung steht.

CPS ist einfach zu verstehen und zu benutzen, führt jedoch zu einigen Nachteilen: Sobald mehrere asynchrone Berechnungen voneinander abhängen, entstehen tief verschachtelte Codestrukturen. Außerdem besitzt der resultierende Quellcode eine gänzlich andere Struktur: Ergebnisse werden nicht mehr zurückgegeben, sondern stets in tiefer verschachtelte Funktionsaufrufe weitergereicht.

Ein anderes sehr gängiges Programmiermuster sind Ereignisse. Auch bei Ereignissen arbeitet der Programmierer nicht mehr mit einfachen Funktionsaufrufen. Stattdessen löst er Ereignisse aus und registriert Ereignis-Handler. Auch dieses Programmiermuster krankt daran, dass der herkömmliche funktionale Datenfluss – also die Transformation von Daten im Programm – unterbrochen wird. Man tut sich bei ereignisbasierten Systemen auch schwer, darin zu erkennen, auf welche Anfrage eine bestimmte Antwort Bezug nimmt.

Funktionen als kombinierbare Transformationen

Funktionen sind einer der einfachsten Bausteine eines Programms: Sie besitzen Parameter als Eingabe und liefern einen Rückgabewert als Ergebnis (Bild 1). Im Idealfall sind sie sogar pur, das bedeutet, dass ihr Ergebnis ausschließlich von ihren Parametern abhängt.

Dieser übersichtliche Aufbau besitzt noch eine weitere wünschenswerte Eigenschaft: Man kann mehrere Funktionen leicht miteinander kombinieren – Composability ist der dafür gängige Terminus. Wie in Bild 2 zu sehen ist, kann man die Funktion *getUser()* sehr leicht mit der Funktion *getMessages()* kombinieren. Es entsteht eine Kette von Transformationen, die mit einer Zeichenkette (hier der Nutzername) beginnt und mit einer Sequenz von Nachrichten endet:

```
let messages = getMessages(getUser("user1"));
```

Funktionale Programmierung baut auf dieser Kombinierbarkeit auf und erlaubt es, Funktionen als simple Bausteine einfach zu neuen, größeren Bausteinen zusammenzubauen:

```
let messagesByUserName = compose(getMessages,getUser);
let messages = messagesByUserName("user1");
```

Typen sind ein hilfreiches Werkzeug, wenn es um funktionale Transformationen geht. JavaScript besitzt natürlich Typen, bietet jedoch noch keine Möglichkeit einer Typnotation, mit der man die Parameter oder Ergebnisse einer Funktion deklarieren könnte. Spätestens seit Facebooks Flow oder Microsofts TypeScript etabliert sich mittlerweile jedoch eine Notation, die auch als Flow Type Notation bekannt ist. Dabei kann man die Typen von Parametern hinter einem Doppelpunkt nach dem Namen angeben:

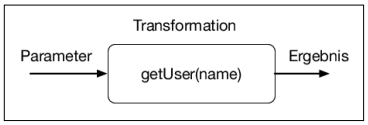
```
function getUser(name : string) {
  ...
}
```

Der Funktions-Rückgabewert lässt sich auch hinter einem Doppelpunkt nach den Parameterklammern deklarieren:

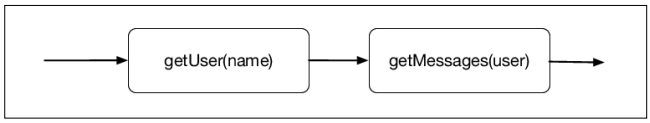
```
function getUser(name : string): User {
  ...
}
```

Bei Typen kann man teilweise auch Typparameter angeben. Beispielsweise kann man bei einem Array-Typ angeben, welchen Typ die Elemente des Arrays besitzen sollen:

```
let users: Array<User> = getUsers();
```



Funktion: Schematische Darstellung einer Funktion (Bild 1)



Funktionskomposition: Schematische Darstellung einer Funktionskomposition (Bild 2)

Man findet diese Typ-Notationen nun immer häufiger in Codebeispielen. In diesem Artikel verwende ich Typen, weil so Ausgangspunkt und Ziel einer Transformation deutlich wird. Notwendig sind Typen für Promises allerdings nicht. Wer lieber herkömmliches JavaScript nutzt, kann diese Typen auch einfach weglassen – sie dienen nur der Dokumentation.

Inversion of Control

Das einfache funktionale Modell wird erschwert, sobald ein Wert wie das User-Objekt nicht mehr direkt (synchron) ermittelt werden kann, sondern nur durch eine asynchrone Anfrage bei einem Dienst. Webentwickler kennen diesen Fall gut: Jede Ajax-Anfrage ist ein solcher asynchroner Zugriff. Meist werden dann Callbacks im CPS verwendet:

```
getUser("user1", (user) => {
  let messages = getMessages(user);
  console.log('${messages.length} Nachrichten
  geladen.');
  ...
});
```

Das Problem: Normalerweise ruft die Anwendung Funktionen auf und erhält Ergebnisse, doch nun reicht die Anwendung Funktionen, also gewissermaßen ein API (Programmierschnittstelle), an das System und wird selbst aufgerufen. Der Steuerungsfluss des Programms wird damit umgekehrt. Das Programm wird zum API und das System übernimmt die Steuerung. Eine solche Funktion lässt sich auch nicht mehr einfach mit anderen Funktionen kombinieren, denn sie liefert kein sinnvoll benutzbares Ergebnis zurück.

Noch schlimmer wird es, wenn im Callback weitere asynchrone Aufrufe verschachtelt sind – also wenn beispielsweise *getMessages()* ebenfalls asynchron wäre:

```
getUser("user1", (user) => {
  getMessages(user, (messages) => {
    console.log('${messages.length} Nachrichten
    geladen.");
  };
});
```

Schnell wird deutlich, wie mit jedem weiteren asynchronen Schritt eine Ebene tiefer verschachtelt wird. Es entsteht die sogenannte Pyramid of Doom, weil die Form der verschachtelten Aufrufe an eine seitlich liegende Pyramide erinnert.

Man kann dennoch leicht zu dem Schluss kommen, dass dieser Code gar nicht so schlimm aussieht. Gerade die ES2015-Arrow-Funktionen erlauben eine recht kompakte Schreibweise. Doch spätestens wenn man eine Callback-Fehlerbehandlung ergänzt, wird deutlich, dass dieses Programmiermuster schnell unübersichtlich wird:

```
getUser("user1", (err, user) => {
  if (err) {
    handleError(err);
  } else {
    getMessages(user, (err, messages) => {
      if (err) {
        handleError(err);
      } else {
      console.log('${messages.length} Nachrichten
      geladen.');
      }
    };
  }
});
```

Dagegen die einfache funktionale Variante mit Fehlerbehandlung:

```
try {
  let messages = getMessages(getUser("user1"));
  console.log('${messages.length} Nachrichten
  geladen.');
} catch (err) {
  handleError(err);
}
```

Asynchrone Programmiermuster mit CPS und der damit einhergehenden Inversion of Control führen also zu unübersichtlicherem, verschachteltem Code, erschwerter Kombinierbarkeit von Programmbausteinen und einer schwerer pflegbaren Fehlerbehandlung.

Promises als Lösung

Promises sind eine Lösung für diese Probleme. Die Idee dahinter ist einfach: Vor der Inversion of Control war die Welt noch in Ordnung. Also kann man versuchen, wieder die einfache Weltsicht synchroner Funktionsaufrufe herzustellen. Dazu bekommen asynchrone Funktionen wieder einen sinnvollen Rückgabewert: ein Stellvertreterobjekt mit dem Namen *Promise*. Dieser steht dabei für einen eventualen Wert – ein Wert, der zwar möglicherweise nicht sofort, aber dennoch irgendwann in Zukunft zur Verfügung stehen wird. Er ist ein Vorab-Stellvertreter für den eigentlichen Wert:

```
let user : Promise<User> = getUser("user1");
user.then(user=>{
});
```

Die Typnotation zeigt, dass die Variable *user* vom Typ *Promise<User>* ist. Das bedeutet, es handelt sich um einen Promise, der einen Wert vom Typ *User* enthalten wird.

Doch auch wenn man einen solchen Stellvertreter-Wert erhalten hat, muss man darauf warten, bis der eigentliche ▶

Wert zur Verfügung steht. Dazu können alle interessierten Parteien den Promise beobachten. Mit der Methode *.then()* des Promise kann man eine Callback-Funktion übergeben, die aufgerufen wird, sobald der Wert des Promise zur Verfügung steht. Also doch wieder Callbacks, mag sich der Laie wundern. Was bringt das Ganze dann?

Die Funktion besitzt nun wieder ein sinnvolles Ergebnis. Dieses kann gespeichert, weitergegeben oder mit anderen Promises kombiniert werden. Weiterhin werden die Callbacks nun am Promise registriert und nicht als Parameter an die asynchrone Funktion übergeben. Die Funktion besitzt also wieder eine Schnittstelle, die einer gewöhnlichen synchronen Funktion entspricht. Übergebene Parameter werden durch die Funktion in einen Promise transformiert.

Das missverstandene .then()

Viele Neulinge, die damit beginnen, Promises einzusetzen, sehen darin vor allem die syntaktische Möglichkeit, Callback-Funktionen mit *then()* zu registrieren, anstatt sie per CPS zu übergeben. Doch oft sieht man deshalb Code, bei dem Promises am Ende nahezu genauso genutzt werden wie asynchrone Funktionen im CPS-Stil:

```
fetch("http://localhost:3000/user").then((response)=>{
  let obj = JSON.parse(response.text);
  doSomethingWith(obj);
  ...
})
fetch("http://localhost:3000/user", (response)=>{
  let obj = JSON.parse(response.text);
  doSomethingWith(obj);
  ...
})
```

Wie man sieht, ist der Code so nahezu identisch. Man hat somit keinerlei Mehrwert aus dem Promiseobjekt gewonnen. Hier wird ignoriert, dass die Methode *then()* nicht nur einfach zum Registrieren von Callbacks gedacht ist. Der eigentliche Zweck der Methode ist es, einen Promise in einen anderen zu transformieren. Der Rückgabewert der übergebenen Callback-Funktion kann dabei wieder ein Promise sein oder ein ganz normaler Wert. Der Rückgabewert der *then()*-Methode ist stets ein Promise, der erfüllt ist, sobald der Rückgabewert des Callbacks erfüllt ist. Damit kann man verschiedene Transformationen von Werten aneinanderhängen und es ist egal, ob die einzelnen Transformationen asynchron oder synchron durchgeführt werden:

```
let userResponse : Promise<HTTPResponse> =
fetch("http://localhost:3000/user");
let responseText : Promise<string> = userResponse.
then(r=>r.text);
let obj:any = responseText.then(JSON.parse);
obj.then(doSomethingWith):
```

Der Code gleicht herkömmlichem synchronem Code. Man muss sich natürlich bewusst sein, dass die Ergebnisse jeweils Promise-Objekte sind. Der *responseText* ist also eben keine Zeichenkette, sondern vom Typ *Promise<string>*. Möchte man daraus die enthaltene Zeichenkette mit *JSON.parse* transformieren, so transformiert man eben einfach das *Promise<string>* per *then()* und *JSON.parse* in ein *Promise <any>*. Der Kenner wird hier eine Parallele von *then()* zu einer anderen bekannten JavaScript-Methode erkennen:

```
let str : string = "{a: 1, b: 2}"
let strInPromise : Promise<string> =
Promise.resolve(str);
let strInArray : Array<string> = [str];
let obj1 : Promise<any> = strInPromise.then(JSON.parse);
let obj2 : Array<any> = strInArray.map(JSON.parse);
```

Die Methode *then()* verhält sich also mit Promises ähnlich wie die Funktion *map()* für Arrays.

Asynchrone Fehler behandeln

Mit Promises kann man Funktionen wieder so nutzen, wie sie gedacht sind: Parameter als Eingabe werden in Werte transformiert. Doch was geschieht, wenn bei der Transformation ein Fehler passiert? Bei herkömmlichen Funktionen kann man Fehlersituationen mit Exceptions behandeln (Listing 1).

Ganz gleich bei welcher dieser Transformationen ein Fehler auftritt – er wird durch den umschließenden *try/catch*-Block gefangen und kann behandelt werden. Wie das im Detail zu geschehen hat, hängt von der Fehlersituation ab.

Dasselbe Programm lässt sich auch mittels Promises verwirklichen. Dabei sind dann bei Bedarf auch asynchrone Transformationen möglich (Listing 2).

Der erste Unterschied zeigt sich in *transformationA():* Damit die Promise-Verarbeitungskette initiiert werden kann, muss das Ergebnis in einen Promise gepackt werden. Das geht jederzeit sehr einfach mit dem Aufruf *Promise.resolve()*.

Listing 1: Exception

```
function transformationA(x:number): number {
  return x + 1;
}
function transformationB(x:number): number {
  throw "Ein Fehler";
  return x + 1;
}
function transformationC(x:number): number {
  return x + 1;
}
try {
  let obj = transformationA(1);
  let obj2 = transformationB(obj);
  let obj3 = transformationC(obj2);
} catch (err) {
  console.error(err);
}
```

Listing 2: Asynchrone Transformationen

```
function transformationA(x : number):
Promise<number> {
  return Promise.resolve(x + 1);
}

function transformationB(x : number):
Promise<number> {
  return new Promise((resolve, reject)=>{
    throw "Ein Fehler";
    return x + 1;
  });
}

function transformationC(x: number): number {
  return x + 1;
}

try {
  let obj = transformationA(1);
  let obj2 = obj.then(transformationB);
  let obj3 = obj2.then(transformationC(obj2));
} catch (err) {
  console.error(err);
}
```

Dieser packt jeden Wert in einen frischen Promise. Wird bereits ein Promise übergeben, dann folgt der resultierende Promise dem Ergebnis des übergebenden. Bei allen anderen Werten ist der Ergebnis-Promise sofort erfüllt.

Auch *transformationB()* wurde im Beispiel als asynchrone Funktion umgeschrieben. Noch bevor der Promise erfüllt werden kann, wird mit einem *throw* eine Exception geworfen. Bei der Ausführung zeigt sich jedoch, dass diese Exception nicht im *catch()*-Block gefangen wird. Das liegt daran, dass der Aufrufstack, auf dem der *catch()*-Block aktiv war, bei der asynchronen Ausführung längst abgebaut ist. Die Funktion, in der die Exception geworfen wurde, wird als Callback in einem anderen Aufrufkontext aufgerufen. Damit wird die Exception jedoch verschluckt und die eigentlich gewünschte Fehlerbehandlung bleibt aus.

Dieses Problem ist keine Eigenart von Promises, sondern eine generelle Herausforderung mit Fehlern in asynchronem Code. Genau dasselbe Dilemma tritt bei asynchronen Funktionen mit Callbacks auf. Doch Promises bieten eine sehr nützliche Lösung für dieses Problem: Mit der Methode *.catch()* kann man an einem Promise einen Callback registrieren, der aufgerufen wird, wenn in der Verarbeitungskette ein Promise fehlschlägt:

```
let obj = transformationA(1);
let obj2 = obj.then(transformationB);
let obj3 = obj2.then(transformationC(obj2));
obj3.catch(err=>{
  console.error(err);
})
```

Mit der *catch()*-Methode lauscht dieses Beispiel auf einen Fehlschlag in der Promise-Verarbeitung. Der Block in Form des *try/catch* ist zwar verschwunden, aber die Promise-Kette von *obj* bis *obj3* ist das Äquivalent dazu. Sie spannt einen Bereich auf, in welchem eine asynchrone Transformation von Werten stattfindet und bei dem Fehler behandelt werden können.

Gefährliche Mischung

Damit die Fehlerbehandlung auf diese Weise zuverlässig funktionieren kann, ist es wichtig, die Promise-Verarbeitungskette aufrechtzuerhalten. Es kann auch schnell zu Problemen führen, wenn man in asynchronen Funktionen asynchrone mit synchronen Bestandteilen mischt:

```
function mischmasch (x) {
  let a = berechne(x);
  return new Promise((resolve, reject)=>{
    return a + 1;
  });
}
```

Wenn in dieser asynchronen Funktion ein Fehler in der synchronen Funktion *berechne(x)* passiert, dann signalisiert die Funktion *mischmasch()* auch einfach den Fehler mit dieser Exception. Asynchrone Funktionen dürfen jedoch ausschließlich entweder erfüllte oder fehlgeschlagene Promises als Ergebnis haben. Ein *throw* außerhalb der Promise-Verarbeitungskette kann nicht als Fehlersituation erkannt werden. Die beste Empfehlung lautet daher, in asynchronen Funktionen die Promise-Verarbeitungskette so früh wie möglich zu beginnen, um zu vermeiden, dass synchroner Code Fehler unkontrolliert verursachen kann. Das Beispiel würde demnach besser so aussehen:

```
function keinmischmasch (x) {
  return new Promise((resolve, reject)=>{
    let a = berechne(x);
    return a + 1;
  });
}
```

Auf diese Weise befindet sich der synchrone Code innerhalb der Promise-Verarbeitungskette und eine Exception wird automatisch zu einem *reject* führen.

Mehrere Promises gleichzeitig

Die bisherigen Beispiele befassten sich stets mit asynchronen Transformationen, die aufeinander aufbauen, infolgedessen voneinander abhängen und deswegen naturgemäß eine feste Reihenfolge implizieren. Bevor es Promises gab, musste man dabei vor allem auf verschachtelte Callback-Aufrufe bauen, sodass Promises an dieser Stelle einen deutlichen Fortschritt bieten. ▶

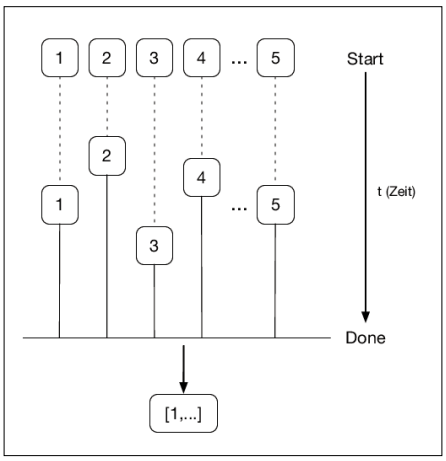
In manchen Fällen muss man für eine Transformation mehrere Quellen zusammenfassen, von denen zumindest einige auch asynchron sein können. Das bedeutet, die einzelnen Promises sind voneinander unabhängig; sie können parallel ausgewertet werden und die Reihenfolge, in der sie aufgelöst werden, spielt keine Rolle. Der ES2015-Standard bietet für diese Zwecke zwei Funktionen: *Promise.all* und *Promise.race*.

Promise.all erhält ein Iterable (zum Beispiel Array) aus Promises und erzeugt einen neuen Promise, der genau dann erfüllt ist, wenn alle eingehenden Promises erfüllt sind (Bild 3). Sobald jedoch auch nur einer der Promises fehlschlägt, wird auch der Sammel-Promise fehlschlagen (Bild 4):

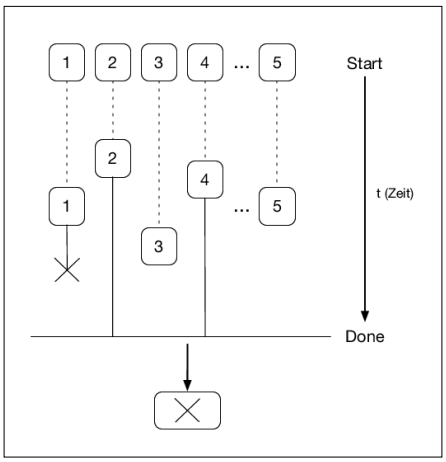
```
function getUserAsync (name:
string): Promise<any> {
  return fetch('http://
  localhost:3000/user/${name}').
  then(r=>JSON.parse(r.text));
}
let users : Array<string> =
["Peter", "Günther"];
function showConsole (obj) {
  console.log(obj);
}
function gruppenAlterAsync(gruppe :
Array<any>): Promise<number> {
  return Promise.all(gruppe).
  then(g=>{
    return g.reduce
    ((sum, u)=>sum + u.alter, 0);
  });
}
let alter = gruppenAlter(users.map(getUserAsync));
alter
.then(showConsole)
.catch(err=>console.error(err));
```

Die Funktion *getUserAsync()* ruft per Ajax Nutzerinformationen ab. Es werden zwei Nutzer abgerufen und das Array der Promises an die Funktion *gruppenAlterAsync()* übergeben. Diese berechnet das Alter und stellt es in einem Promise bereit. Doch was, wenn die Nutzerobjekte bereits vorliegen?

```
let users = [
  { name: "Peter", alter: 30 },
  { name: "Günther", alter 32 }
];
let alter = gruppenAlterAsync(users);
alter
.then(showConsole)
.catch(err=>console.error(err));
```



Promise.all erhält ein Iterable (zum Beispiel ein Array) aus Promises (Bild 3)



Fehlschlag bei *Promise.all()* beziehungsweise *Promise.race()* (Bild 4)

Wie man sieht kommt *Promise.all* (und damit *gruppenAlterAsync)* einfach damit zurecht, die Objekte direkt und nicht als Promise-Objekte zu erhalten. *Promise.all* wandelt bei Bedarf die übergebenen Objekte in erfüllte Promises um. Damit ist das auf Promises basierende Async-API sowohl für asynchrone als auch direkt vorliegende Objekte nutzbar.

Die Funktion Promise.race()

Die zweite Funktion – *Promise.race()* – funktioniert sehr ähnlich wie *Promise.all()*. Der essenzielle Unterschied: Der Sammel-Promise ist erfüllt, sobald der erste Eingabepromise erfüllt ist (Bild 5). Das bedeutet auch, dass nicht alle übergebenen Promises erfüllt sein müssen – was ein wichtiger Aspekt gegenüber *Promise.all* ist.

Bei einem Fehlschlag verhält sich die Funktion wie *Promise.all()*: Der erste Fehlschlag führt zum Ablehnen des Sammel-Promise. Doch wozu ist dieses Verhalten nützlich?

Ein typischer Anwendungsfall für *Promise.race()* ist die Kombination einer asynchronen Funktion mit einem Timeout:

```
function timeout (time) {
  return new Promise
  ((resolve, reject)=>{
    setTimeout(()={
    reject();
    },time);
  });
}
function getUserAsyncWithTimeout(name) {
  return Promise.race([
    getUserAsync(name),
    timeout(3000)
  ]);
}
```

Die Funktion *timeout()* erwartet als Parameter die Zeit, nach der der von ihr erzeugte Promise fehlschlagen soll. Erfüllt wird es nie. Auch das Caching eines Default-Value nach einer wählbaren Verzögerung lässt sich so verwirklichen:

```
function delayedValue (value, time) {
  return new Promise((resolve, reject)=>{
    setTimeout(()={
```

```
      resolve(value);
    },time);
  });
}
function getUserAsyncWithDefault(name, default) {
  return Promise.race([
    getUserAsync(name),
    delayedValue(default, 3000)
  ]);
}
```

Die Funktion *getUserAsyncWithDefault()* ruft mit *getUserAsync()* – also einem Ajax-Request – einen Nutzer ab. Gleichzeitig wird jedoch ein Promise erzeugt, das nach einer Zeitspanne von 3000 ms mit einem festgelegtem Default-Wert erfüllt wird. Je nachdem, welcher der beiden Promises schneller erfüllt wird, gewinnt entweder der Wert vom Server oder der lokal als Default vorgehaltene Wert. Auch ganz andere Anwendungen sind realisierbar:

```
function clickPromise (view) {
  return new Promise((resolve, reject)=>{
    view.once("click", ()=>resolve(view);
  });
}
function yesNoDialog () {
  let yesButton = new Button();
  let cancelButton = new Button();
  return Promise.race([
    clickPromise(saveButton),
    clickPromise(cancelButton)
  ]).then(b=>b===yesButton);
}
```

In diesem ungewöhnlicheren Beispiel wird ein Dialog mit zwei Buttons erzeugt. Die Klick-Ereignis-Handler der Buttons werden mit Promises verknüpft. Sobald einer der beiden Buttons angeklickt wird, ist der Dialog-Promise erfüllt und liefert einen Wahrheitswert abhängig davon, ob der Yes-Button oder der No-Button geklickt wurde. Natürlich hätte man diesen Dialog auch mit einem Promise implementieren können, bei dem dessen *resolve()*-Funktion je nach Klickereignis entweder mit einem *true* oder einem *false* erfüllt wird, aber dieses Beispiel soll gerade die Kombinierbarkeit von Promises auch bei ereignisbasierten Quellen widerspiegeln.

Promise Reflection

Die von *Promise.all* und *Promise.race* erzeugten Promises schlagen fehl, sobald auch nur einer der übergebenen Promises fehlschlägt. Dieses Verhalten ist nicht immer gewünscht. Mit einer kleinen Hilfsfunktion kann man Abhilfe schaffen: Die Funktion transformiert einen gegebenen Promise in einen Promise, dessen Wert den Erfüllt- oder Fehler-Zustand des ursprünglichen Promise beschreibt:

```
function reflect (promise) {
  return promise.then(
    v=>({ state: "fullfilled", value: v}),
    e=>({ state: "rejected", error: e})
  );
}
```

Mit dieser Funktion lässt sich eine Variante von *Promise.all* implementieren, die nicht beim ersten fehlgeschlagenen Promise den resultierenden Promise fehlschlagen lässt, sondern genau dann erfüllt ist wenn alle übergebenen Promises entweder erfüllt oder fehlgeschlagen sind:

```
function settle (promises) {
  return Promise.all(promises.map(reflect));
}
```

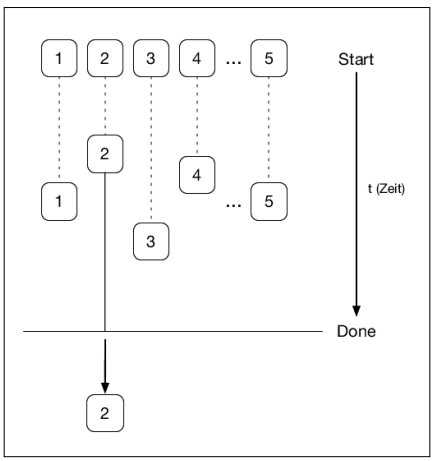
Promises sind Werte, und als solche eignen sie sich deshalb besonders für funktionale Anwendungen. Man kann sie jedoch auch nutzen, um asynchrone Seiteneffekte zu steuern. Ein typischer Fall dafür ist DOM-Manipulation, denn das imperative DOM-API funktioniert über Operationen, die das DOM mit Seiteneffekten manipulieren. Weitere Anwendungsfälle findet man auch bei Node.js, beispielsweise mit Zugriffen auf das Dateisystem: Eine bestimmte Abfolge von asynchronen Schreibvorgängen soll zum Beispiel in genau dieser Reihenfolge durchgeführt werden, und es sollen dabei auch keine Zugriffe gleichzeitig erfolgen.

Würde man die einzelnen Schreibvorgängen direkt durchführen, entstünde ein Array von Promises. Die Schreibvorgänge würden sofort begonnen und parallel ablaufen. Die Zugriffe würden sich möglicherweise zeitlich überlappen, oder sie würden in einer anderen Reihenfolge ausgeführt werden, als eigentlich notwendig. Eigentlich möchte man, dass jeder Schreibvorgang erst beginnt, wenn der Vorgänger erfolgreich abgeschlossen wurde.

Um dieses Problem zu lösen, werden all diese asynchronen Funktionsaufrufe in sogenannte Thunks gepackt:

```
let arrayOfRecords = [...];
function writeToFileAsync (record) {
  ...
}
let effects = arrayOfRecords.map(r=>
()=>writeToFileAsync(r));
```

Auf diese Weise werden die Schreibvorgänge nicht sofort ausgeführt, sondern es entsteht für jeden Vorgang eine Funktion ohne Parameter, und erst deren Aufruf startet auch den Vorgang. Wenn man nun einfach jede Funk- ▶

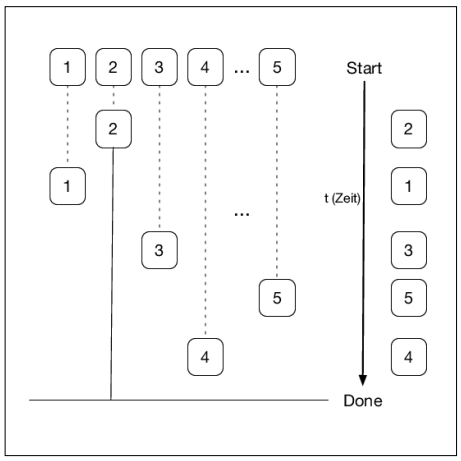


Promise.race() funktioniert sehr ähnlich wie *Promise.all()* (Bild 5)

tion in dieser Reihenfolge aufruft, dann werden zwar alle Vorgänge initiiert, aber die Reihenfolge der Abarbeitung ist nicht definiert:

```
Promise.all(effects.map(e=>e())
.then(()=>console.log("done"));
```

Dank *Promise.all()* kann man tatsächlich die erfolgreiche Abarbeitung aller Schreibvorgänge abwarten, aber dennoch hat man auf diese Weise keine Kontrolle darüber, in welcher Reihenfolge die Effekte angewandt werden. Die folgende Funktion implementiert das gewünschte Verhalten:



Promise.all: Seiteneffekte nicht serialisiert (Bild 6)

```
function series(effects) {
  return effects.reduce(
    (promise, effect)=>promise.then(()=>effect()),
    Promise.resolve()
  );
}
```

Sie erwartet ein Array aus Thunks. Statt dann jedoch das Array an *Promise.all()* zu übergeben, initiiert man zuerst einen einzelnen, leeren, erfüllten Promise mit *Promise.resolve()* und gibt ihn als Initial-Wert an die *reduce()*-Methode. Damit wird die benötigte Promise-Kette initiiert. *Reduce* wertet dann stets einen Thunk nach dem anderen aus.

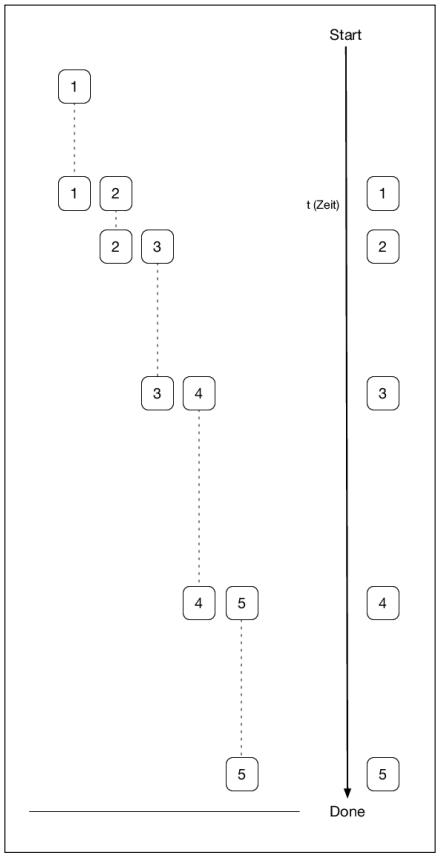
Das Ergebnis des jeweiligen Thunks ist stets ein neuer Promise, der in die nächste Iteration weitergereicht wird. Auf diese Weise wird der jeweils nächste Thunk erst ausgeführt, wenn der Promise des vorherigen Effekts erfüllt ist. Im folgenden Beispiel wird eine Funktion *delayedEffect()* eingeführt, die eine Funktion mit Seiteneffekt nach einer übergebenen Verzögerung ausführt und danach den Promise erfüllt. Das Array *effects* enthält Thunks mit Aufrufen dieser Funktion und unterschiedlich durchmischten Verzögerungszeiten. Die Funktion *series* zwingt die einzelnen Effekte in eine feste Reihenfolge und sorgt dafür, dass sich die Ausführung nicht überlappen kann:

```
function delayedEffect (delay,
effect) {
  return new Promise
  ((resolve,reject)=>{
    setTimeout(()=>{
      effect();
      resolve();
    }, delay);
  });
}
function log (msg) {
  return ()=>console.log(msg);
}
var effects = [
  ()=>delayedEffect(300,
  log("First")),
  ()=>delayedEffect(100,
  log("Second")),
  ()=>delayedEffect(500,
  log("Third")),
  ()=>delayedEffect(1000,
  log("Fourth")),
  ()=>delayedEffect(700,
  log("Fifth")),
];
series(effects);
```

Bild 6 zeigt, wie die Effekte mit *Promise.all* ausgeführt werden, und Bild 7 zeigt die serielle Verarbeitung.

Ein Ausblick: Async/Await

Promises sind für viele Entwickler ungewohnt oder gar umständlich, weil man stets daran denken muss, die Ergebnisse asynchroner Funktionen in Promises zu packen und jeden Zugriff auf so verpackte Ergebnisse innerhalb einer Promise-Kette mit *.then()* zu realisieren.



Alternative: Seiteneffekte serialisiert (Bild 7)

Microsoft hat in seinem .NET Framework eine Notation erfunden, die diese Idiome mit einer hübschen Syntax aufbereitet. Es gibt einen Vorschlag, eine solche Syntax für eine der kommenden Standardversionen von ECMAScript umzusetzen:

```
async function getUserAsync (name) {
  return fetch(name);
}
async function addiereAlter(name1,
name2) {
  let user1 = await getUserAsync
  (name1);
  let user2 = await getUserAsync
  (name2);
  return user1.alter + user2.alter;
}
```

Jede asynchrone Funktion wird mit dem neuen Schlüsselwort *async* gekennzeichnet. Die Funktion kümmert sich dann selbst darum, dass jedes per *return* zurückgegebene Ergebnis auto-

matisch in einen Promise gepackt wird. Für den Nutzer sieht die Funktion *addiereAlter()* so aus, als ob sie eine Zahl zurückgibt.

Innerhalb dieser asynchronen Funktionen kann man mit dem Schlüsselwort *await* vor einem Funktionsaufruf auf das asynchron abgerufene Ergebnis warten. Natürlich blockiert JavaScript nicht wirklich an dieser Position, sondern der Code wird so umgeschrieben, dass er später an dieser Position wieder fortgesetzt werden kann. Da der JavaScript-Transpiler Babel bereits *async/await* unterstützt, kann man begutachten, wie der obige Code in älteren JavaScript-Engines einsetzbar ist:

```
var getUserAsync = function () {
  var ref = _asyncToGenerator(function*(name) {
    return fetch(name);
  });
  return function getUserAsync(_x) {
    return ref.apply(this, arguments);
  };
}();
```

Der Code der asynchronen Funktion *getUserAsync()* wird in eine Generator-Funktion gesteckt, die dann stattdessen aufgerufen wird. Diese Generator-Funktion wird mit einer Hilfsfunktion *_asyncToGenerator()* umgewandelt (Listing 3).

Der Code dieser Hilfsfunktion sieht kompliziert aus – insbesondere wenn man mit Generatoren nicht vertraut ist. Die Hilfsfunktion erzeugt eine Funktion, die intern die Generatorfunktion aufruft und damit einen Generator erzeugt. Dann gibt sie ein Promise zurück, dessen Ausführungsfunktion den Generator nutzt. Im Fall der *getUserAsync()* wird damit einfach ein Promise erzeugt, der auf das Promise-Ergebnis von *fetch()* lauscht. Damit kann man diese Funktion natürlich auch ohne *async/await* benutzen.

Der Nutzen des Generators wird erst in der Funktion *addiereAlter()* deutlich:

```
var addiereAlter = function () {
  var ref = _asyncToGenerator(function*(name1, name2) {
    var user1 = yield getUserAsync(name1);
    var user2 = yield getUserAsync(name2);
    return user1.alter + user2.alter;
  });
  return function addiereAlter(_x2, _x3) {
    return ref.apply(this, arguments);
  };
}();
```

Hier sieht man, dass die *await*-Aufrufe in *yield*-Ausdrücke umgesetzt werden. Auf diese Weise wird klar, wie die asynchrone Funktion auf die *awaits* warten kann: Der Generator arbeitet die Generator-Funktion bis zum *yield* ab, liefert dann wie gewohnt ein Ergebnis. Bei einem erneuten Aufruf setzt er wieder auf und arbeitet die Generatorfunktion weiter ab. Wenn eine JavaScript-Engines noch keine Generatoren unterstützt, kann Babel auch mittels Facebooks Regenerator Code generieren, der diese implementiert.

Listing 3: Generator-Funktion

```
function _asyncToGenerator(fn) {
  return function () {
    var gen = fn.apply(this, arguments);
    return new Promise(function (resolve, reject) {
      function step(key, arg) {
        try {
          var info = gen[key](arg);
          var value = info.value;
        } catch (error) {
          reject(error);
          return;
        }
        if (info.done) {
          resolve(value);
        } else {
          return Promise
            .resolve(value).then(function (value) {
              return step("next", value);
            }, function (err) {
              return step("throw", err);
            });
        }
      }

      return step("next");
    });
  };
}
```

Fazit

Möchte man Promises intuitiv verstehen, ist es wichtig, sie als einfache Werte zu verstehen und eben gerade nicht als ein spezielles asynchrones API. Ist ein Promise einmal erfüllt, dann ändert sich der enthaltene Wert nicht mehr. Die Methode *.then()* dient dazu, bestimmte Promise-Werte in andere zu transformieren. Die Stärke des Promise-Konzepts liegt vor allem in der Kombinierbarkeit. Entwickler, die funktionale Programmierkenntnisse besitzen, können ihr Wissen mit Promises besser und zielführender einsetzen als beim ansonsten auch häufig anzutreffendem Continuation Passing Style. ■

Jochen H. Schmidt
ist als Autor, Berater und Software-Entwickler tätig. Schwerpunkte seiner Aktivitäten sind Webentwicklung und Webtechnologien. Er ist Verfasser von bekannten Fachbüchern zum Thema Common Lisp.

DAS FILE API

Zugriff auf Dateien

Das File API ermöglicht den Zugriff auf Dateien per JavaScript.

Über das File API ist es möglich, per JavaScript auf lokale Dateien des Nutzers zuzugreifen, sofern dieser die jeweiligen Dateien zuvor in einem entsprechenden Dialog ausgewählt hat. Die Spezifikation unter https://www.w3.org/TR/FileAPI definiert dabei folgende Interfaces:

- Das Interface *File* repräsentiert eine einzelne Datei und enthält Informationen wie den Namen der Datei oder das letzte Änderungsdatum.
- Das Interface *FileList* repräsentiert eine Liste von *File*-Objekten, die vom Nutzer ausgewählt wurden.
- Das Interface *Blob* repräsentiert binäre Daten.
- Das Interface *FileReader* stellt Methoden zur Verfügung, um Objekte vom Typ *File* oder vom Typ *Blob* zu lesen.

Mittlerweile unterstützen so gut wie alle aktuellen Browser das File API (Bild 1).

Damit über das File API auf lokale Dateien zugegriffen werden kann, muss der Nutzer zunächst die entsprechenden Dateien auswählen. Dies stellt sicher, dass nicht unbemerkt per JavaScript beliebige Dateien gelesen werden können.

Für das Auswählen von Dateien gibt es prinzipiell zwei Möglichkeiten: zum einen den schon erwähnten Dateidialog, zum anderen besteht aber auch die Möglichkeit, Dateien per

Das File API wird von fast allen Browsern unterstützt (Bild 1)

Listing 1: Dateien auswählen über Dateidialog

```
function handleFileSelected(event) {
  let files = event.target.files;
  let output = '';
  for (let i = 0; i < files.length; i++) {
    let file = files[i];
    output += '<li>' + '<strong>' + file.name +
    '</strong>' + ' (' + (file.type || "n/a") + ') -
    ' + file.size + 'bytes' ' + ' geändert am: ' +
    file.lastModifiedDate.toLocaleDateString() +
    '</li>';
  }
  document.getElementById('list').innerHTML = '<ul>'
  + output + '</ul>';
}
document.getElementById('files').addEventListener
('change', handleFileSelected, false);
```

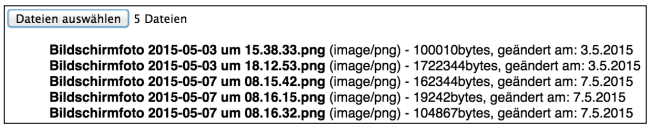
Drag and Drop in einen bestimmten Bereich einer Webseite zu ziehen.

Auswählen von Dateien per Dateidialog

Einen Dateidialog definiert man bekanntermaßen über das *<input>*-Element, wobei man dessen *type*-Attribute auf den Wert *file* setzt:

```
<input type="file" id="files" multiple />
```

Über das boolesche Attribut *multiple* lässt sich zudem steuern, ob es möglich ist, in dem Dateidialog mehrere Dateien auszuwählen. Wählt der Nutzer nun eine oder mehrere Dateien aus, wird im Hintergrund das *change*-Event für das



Die ausgewählten Dateien werden in einer HTML-Liste angezeigt (Bild 2)

<input>-Element ausgelöst. Um auf Dateiauswahlen reagieren zu können, muss man daher einfach wie in Listing 1 gezeigt einen Event Listener für dieses Event registrieren.

An die selektierten Dateien gelangt man innerhalb eines Event Listeners über die *files*-Eigenschaft des Elements, für das das Event ausgelöst wurde *(event.target.files)*. In dieser Liste sind alle selektierten Dateien als File-Objekte enthalten, die zwar noch nicht den Inhalt der jeweiligen Datei bereitstellen, aber bereits allgemeine Informationen wie den Dateinamen (Eigenschaft *name),* den Dateityp (Eigenschaft *type),* die Dateigröße (Eigenschaft *size)* und das letzte Änderungsdatum (Eigenschaft *lastModifiedDate)*. Im Beispiel werden alle diese Informationen unterhalb der Schaltfläche zur Dateiauswahl in den DOM-Baum eingebaut (Bild 2).

Auswählen von Dateien per Drag and Drop

Um eine Datei per Drag and Drop auszuwählen, muss man zunächst innerhalb der entsprechenden Webseite einen Bereich definieren, der als Ziel für die Drag-and-Drop-Operation verwendet werden soll (im Weiteren Drop-Ziel genannt). Im Beispiel ist dies ein * *-Element mit der ID *file-selection-drop-target*.

An diesem Element werden anschließend zwei Event Listener registriert: der Event Listener für das *dragover*-Event *(handleDragOver())* wird ausgeführt, wenn eine Datei über das Element gezogen (aber noch nicht losgelassen) wird, der Event Listener für das *drag*-Event *(handleFileSelected())* ▶

Listing 2: Dateien auswählen per Drag and Drop

```
function handleFileSelected(event) {
  event.stopPropagation();
  event.preventDefault();
  let files = event.dataTransfer.files;
  let output = '';
  for (let i = 0; i < files.length; i++) {
    let file = files[i];
    output += '<li>' + '<strong>' + file.name +
    '</strong>' + ' (' + (file.type || "n/a") + ') -
    ' + file.size + 'bytes, ' + ' geändert am: ' +
    file.lastModifiedDate.toLocaleDateString() +
    '</li>';
  }
  document.getElementById('list').innerHTML =
  '<ul>' + output + '</ul>';
}
```

```
function handleFileDraggedOver(event) {
  event.stopPropagation();
  event.preventDefault();
  event.dataTransfer.dropEffect = 'copy';
}

let dropTarget = document.getElementById('target');
dropTarget.addEventListener(
  'dragover',
  handleFileDraggedOver,
  false
);
dropTarget.addEventListener(
  'drop',
  handleFileSelected,
  false
);
```

Tabelle 1: Methoden von FileReader

Methode	Beschreibung
readAs BinaryString()	Liest die Daten als binäre Zeichenfolge ein
readAsText()	Liest die Daten als Text ein
readAs DataURL()	Liest die Daten als Daten-URL ein
readAs ArrayBuffer()	Liest die Daten als Array-Buffer ein
abort()	Bricht den Lesevorgang ab

Tabelle 2: Event Handler für FileReader

Eigenschaft	Beschreibung
onabort	Handler für das *abort*-Ereignis, das ausgelöst wird, wenn der Lesevorgang abgebrochen wurde
onerror	Handler für das *error*-Ereignis, das ausgelöst wird, wenn ein Fehler beim Lesevorgang aufgetreten ist
onload	Handler für das *load*-Ereignis, das ausgelöst wird, wenn der Lesevorgang erfolgreich abgeschlossen wurde
onloadstart	Handler für das *loadstart*-Ereignis, das ausgelöst wird, wenn der Lesevorgang gestartet wurde
onloadend	Handler für das *loadend*-Ereignis, das ausgelöst wird, wenn der Lesevorgang abgeschlossen wurde
onprogress	Handler für das *progress*-Ereignis, das während des Lesevorgangs ausgelöst wird, um über den Fortschritt zu informieren

wird ausgeführt, wenn die Datei dann über dem Element losgelassen wurde.

Innerhalb des Event Listeners *handleDragOver()* kann beispielsweise wie in Listing 2 über *evt.dataTransfer.dropEffect* für den Nutzer durch ein entsprechendes Symbol gekennzeichnet werden, um welche Art von Drag-and-Drop-Operation es sich handelt. Das eigentliche Selektieren der Dateien geschieht innerhalb des Event Listeners *handleFileSelected()*, wobei hier wie auch schon zuvor bei der Auswahl über den Dateidialog auf die gleiche Weise auf die Dateiinformationen zugegriffen wird.

Lesen von Dateien

Bisher haben wir die Dateien nur ausgewählt, aber noch nicht deren Inhalt gelesen. Um den Inhalt von Dateien zu lesen, benötigt es nämlich zunächst eine Instanz von FileReader. Dieser Typ stellt verschiedene Methoden für das Lesen von Dateien zur Verfügung (Tabelle 1). Im Einzelnen sind dies die Methode *readAsText()* für das Lesen einer Datei als Zeichenkette, *readAsBinaryString()* für das Lesen einer Datei als binäre Zeichenkette, *readAsDataURL()* für das Lesen einer Datei als Data-URL und *readAsArrayBuffer()* für das Lesen einer Datei als Array-Buffer.

Ruft man eine dieser Methoden auf, wird man nach erfolgreichem Einlesen der Datei über das *load*-Event benachrichtigt – einen entsprechend zuvor definierten Event Handler oder registrierten Event Listener vorausgesetzt. Wurde das *load*-Event ausgelöst, steht zudem der Inhalt der Datei über die Eigenschaft *result* (ebenfalls an der entsprechenden FileReader-Instanz) zur Verfügung.

Ein Beispiel für das Lesen von Dateien zeigt Listing 3. Für jede durch den Nutzer ausgewählte Datei wird anhand des Dateityps überprüft, ob es sich um eine Textdatei oder um eine Bilddatei handelt (andere Fälle werden nicht betrachtet).

Im Fall einer Textdatei wird der Inhalt über *readAsText()* gelesen und dem DOM-Baum hinzugefügt. Im Fall einer Bilddatei wird der Inhalt über *readAsDataURL()* als Data-URL gelesen und ein *<img>*-Element mit einer Thumbnail-Ansicht des Bildes erzeugt. Zusätzlich zum *load*-Event beziehungsweise dem entsprechenden *onload*-Event Handler stehen ei-

Listing 3: Lesen von Text- und Bilddateien

```
function handleFileSelected(event) {
  let files = event.target.files;
  for (let i = 0; i < files.length; i++) {
    let file = files[i];
    let reader = new FileReader();
    if(file.type.match('text.*')) {
      reader.onload = (event) => {
        let span = document.createElement('span');
        span.innerHTML = reader.result;
        document.getElementById('list').
        insertBefore(span, null);
      };
      reader.readAsText(file);
    } else if(file.type.match('image.*')) {
      reader.onload = (event) => {
        let span = document.createElement('span');
        span.innerHTML = '<img class="thumbnail"
        src="' + reader.result + '"/>';
        document.getElementById('list').
        insertBefore(span, null);
      };
      reader.readAsDataURL(file);
    }
  }
}

document.getElementById('files').addEventListener
('change', handleFileSelected, false);
```

Listing 4: Den Lesefortschritt überwachen

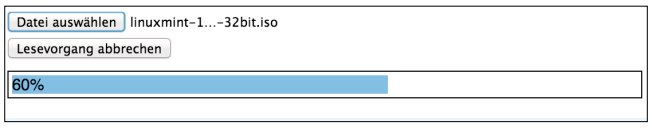
```
let progress = document.querySelector('.percent');
let reader = new FileReader();
function updateProgress(event) {
  if (event.lengthComputable) {
    let percentLoaded = Math.round(
      (event.loaded / event.total) * 100
    );
    if (percentLoaded < 100) {
      progress.style.width = percentLoaded + '%';
      progress.textContent = percentLoaded + '%';
    }
  }
}
function handleError(event) {
  switch (event.target.error.code) {
    case event.target.error.NOT_FOUND_ERR:
      console.error('Datei wurde nicht gefunden');
      break;
    case event.target.error.NOT_READABLE_ERR:
      console.error('Datei konnte nicht
      gelesen werden');
      break;
    case event.target.error.ABORT_ERR:
      break;
    default:
      console.error('Unbekannter Fehler');
  }
  ;
}
function handleFileSelect(event) {
  progress.style.width = '0%';
  progress.textContent = '0%';
  reader = new FileReader();
  reader.onprogress = updateProgress;
  reader.onerror = handleError;
  reader.onabort = (event) => {
    console.log('Lesen der Datei abgebrochen');
  };
  reader.onloadstart = (event) => {
    document.getElementById('progress-bar')
      .className = 'loading';
  };
  reader.onload = (event) => {
    progress.style.width = '100%';
    progress.textContent = '100%';
  };
  reader.readAsBinaryString(event.target.files[0]);
}
document.getElementById('files').
addEventListener('change', handleFileSelect, false);
function abortRead() {
  reader.abort();
}

document.getElementById('abort-file-reading').
addEventListener('click', abortRead, false);
```

nige weitere Event Handler (beziehungsweise entsprechende Events) zur Verfügung, die in Tabelle 2 aufgelistet sind.

Den Lesefortschritt überwachen

Besonders beim Lesen von großen Dateien, das einige Zeit in Anspruch nehmen kann, ist es sinnvoll, Nutzern den Fortschritt des Lesens mitzuteilen. Dazu bietet das File API beziehungsweise der Typ *FileReader* den Event Handler *onprogress* an. Das entsprechende Event-Objekt, das hierbei versendet wird, ist vom allgemeinen Typ *ProgressEvent* (https://www.w3.org/TR/progress-events) und enthält unter anderem die beiden Eigenschaften *loaded* und *total*, die in diesem Fall die Anzahl an geladenen Bytes und die komplette Anzahl an Bytes für die jeweilige Datei enthalten.

Der aktuelle Fortschritt lässt sich dann mit Hilfe dieser beiden Eigenschaften ganz einfach ermitteln, wie in Listing 4 zu sehen ist. Bild 3 zeigt einen Screenshot der Anwendung. Als Beispiel wurde hier eine mehrere Hundert MByte große ISO-Datei verwendet; entsprechend wird im Code die Methode *readAsBinaryString()* benutzt, weil es sich bei der ISO-Datei um eine binäre Datei handelt.



Über den Fortschrittsbalken wird der Lesefortschritt angezeigt (Bild 3)

Fazit

Über das File API lassen sich sowohl Text- als auch Binärdateien lesen, die zuvor vom Nutzer über einen Dateidialog oder per Drag and Drop ausgewählt wurden. Ein Zugriff auf andere Dateien ist dagegen aus Sicherheitsgründen nicht möglich. Das File API wird von so gut wie allen aktuellen Browsern unterstützt. ■

Philipp Ackermann
arbeitet beim Fraunhofer-Institut für Angewandte Informationstechnologie FIT an Tools zum teilautomatisierten Testen von Web Compliance und ist Autor zweier Fachbücher über Java und JavaScript.
http://philipackermann.de

REACT-NATIVE-BIBLIOTHEKEN

Adieu WebView

React-Native-Bibliotheken stellen React für iOS- und Android-Apps zur Verfügung.

Auf PhoneGap basierende Applikationen sind im Grunde genommen Webseiten, die mit einem hauseigenen Browser ausgeliefert werden. React Native dagegen nutzt einen JavaScript-Interpreter samt Bindings zum Hostbetriebssystem – eine Architektur, die mehr Flexibilität verspricht.

Der erste und wichtigste Vorteil ist, dass das Benutzerinterface der erzeugten Applikationen dem des Betriebssystems eins zu eins entsprechen kann: Die Runtime mappt die vom Entwickler angegebenen Tags zur Laufzeit auf native Steuerelemente, womit GUI-Stacks wie jQuery UI oder Kendo unnötig werden. Vorteil zwei ist die einfache Erweiterbarkeit. Implementiert ein natives Element die Interfacespezifikation der Runtime, so ist es für JavaScript ansprechbar.

Facebook entwickelte das Framework anfangs selbst, um es vor einigen Monaten in den Markt zu werfen. Seit dem etwas unrund verlaufenen Start ging reichlich Zeit ins Land. Es lohnt sich, das mittlerweile stabil arbeitende Produkt genauer unter die Lupe zu nehmen.

React Native installieren

React Native wird auf Seiten der Workstation durch ein Kommandozeilenwerkzeug realisiert, das in einer Node.js-Ausführungsumgebung lebt (Version 4). Prüfen Sie daher im ersten Schritt, ob eine beziehungsweise welche Version der Ausführungsumgebung auf Ihrer Maschine vorhanden ist.

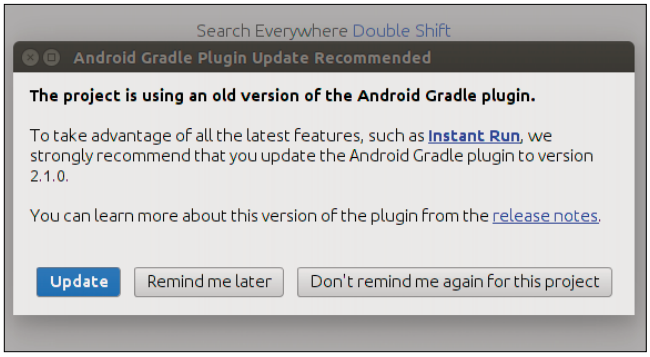
Enthält Ihre Workstation schon eine veraltete oder eine zu neue Version, so müssen Sie diese deinstallieren. Im nächsten Schritt erfolgt das Herunterladen aus einem speziellen Repository, das nur Versionen der Bauart 4.x anbietet:

```
sudo apt-get install -y build-essential
curl -sL https://deb.nodesource.com/setup_4.x | sudo -E
bash -
sudo apt-get install -y nodejs
sudo ln -s /usr/bin/nodejs /usr/bin/node
```

Das eigentliche Herunterladen von React Native – das Kürzel CLI steht in der Node.js-Welt für Command Line Interface – erfolgt sodann mit dem Paketmanager NPM. Je nach Konfiguration Ihres Systems kann es notwendig sein, den Aufruf von NPM als Superuser auszuführen:

```
tamhan@TAMHAN14:~$ npm install -g react-native-cli
```

Facebook empfiehlt in der offiziellen Dokumentation zudem das Deployment von Watchman. Es handelt sich dabei um ein kleines Werkzeug, das Dateisystemänderungen automatisch erkennt.



React Native hinkt der Weiterentwicklung von Android Studio mitunter etwas hinterher (Bild 1)

Die Watchman-Installation erfolgt ebenfalls mittels NPM, setzt allerdings prinzipiell Superuser-Rechte voraus:

```
tamhan@TAMHAN14:~$ sudo npm install -g watchman
```

Achten Sie darauf, ob das Programm im Rahmen der Installation Warnungen ausgibt. Steht am Ende keine mit Error beginnende Zeile, so war die Installation erfolgreich.

Die für die Erzeugung von Android-Applikationen notwendigen Werkzeuge müssen separat heruntergeladen werden. Der einfachste und komfortabelste Weg dazu ist das Installieren einer Vollversion von Android Studio. Wer die IDE noch nicht hat, kann sie unter https://developer.android.com/studio herunterladen. Starten Sie die IDE nach der erfolgreichen Installation und öffnen Sie den in der Toolbar verlinkten SDK Manager. Für die Arbeit mit React Native sind laut Facebook die folgenden Pakete vorgesehen:

- Android 6.0 (Marshmallow),
- Google APIs,
- Intel x86 Atom System Image,
- Intel x86 Atom_64 System Image,
- Google APIs Intel x86 Atom_64 System Image,
- SDK Tools,
- Android SDK Build-Tools 23.0.1 I.

Da der Autor seine Programme ausschließlich auf echter Hardware testet, sind auf seiner Workstation die diversen Images nicht vorhanden. Für die folgenden Schritte stellte dies kein Problem dar. Es wäre allerdings vorstellbar, dass Supportmitarbeiter von Facebook beim Melden technischer Probleme diese Situation monieren.

Zu guter Letzt muss die Umgebungsvariable *ANDROID_HOME* gesetzt werden. Sie ermöglicht dem CLI das Finden

des Android-SDK. Öffnen Sie die für die Konfiguration der Shell zuständige Datei *~/.bashrc* in einem Editor Ihrer Wahl. Ich nutze dafür gerne gedit, der sich in der Kommandozeile durch Eingabe von *gedit ~/.bashrc* aktivieren lässt. Fügen Sie sodann folgende Zeile ein, wobei das Verzeichnis natürlich durch das auf Ihrer Workstation gültige zu ersetzen ist:

```
#Pfad zum Android-SDK
export ANDROID_HOME=/home/tamhan/Android/Sdk
```

Nach einem Neustart des Terminals sollte die Umgebungsvariable am Platz sein. Dies lässt sich durch Nutzung des *echo*-Befehls prüfen. Er gibt bei korrekter Parametrierung den Wert der Umgebungsvariable aus:

```
tamhan@TAMHAN14:~/$ echo
$ANDROID_HOME
/home/tamhan/Android/Sdk
```

Nach diesen einführenden Schritten ist es an der Zeit, ein Projektskelett zu generieren. Dabei wartet eine kleine Falle auf offline arbeitende Entwickler. Auch wenn die CLI selbst lokal auf unserer Workstation installiert ist: Das Erzeugen eines neuen Projektskeletts erfolgt durch das Herunterladen eines Archivs aus dem Internet, das anschließend auf Ihrer Workstation ausgepackt und ausgeführt wird. Der dazu notwendige Code sieht – inklusive der Erzeugung eines Arbeitsverzeichnisses – so aus:

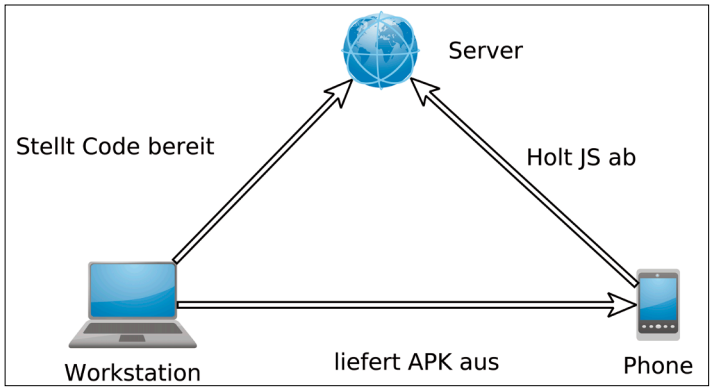
```
tamhan@TAMHAN14:~$ mkdir workspaceReact
tamhan@TAMHAN14:~$ cd workspaceReact/
tamhan@TAMHAN14:~/workspaceReact$ react-native init
NMGTest1
...
To run your app on Android:
Have an Android emulator running (quickest way to get
started), or a device connected
cd /home/tamhan/workspaceReact/NMGTest1
react-native run-android
tamhan@TAMHAN14:~/workspaceReact$
```

Wundern Sie sich nicht, wenn dieser Prozess einige Zeit in Anspruch nimmt. Auch auf der achtkernigen Workstation des Autors waren Wartezeiten von bis zu fünf Minuten keine Seltenheit Wechseln Sie sodann in das Projektunterverzeichnis und geben Sie dessen Struktur mittels *Tree* aus.

Verzeichnisstruktur

React-Native-basierte Projekte bestehen aus einer Gruppe von Unterprojekten: Der Treedump zeigt die einzelnen Verzeichnisse. Im Ordner */android/* findet sich ein vollständiges Android-Projekt, das sich auf Wunsch mit Android Studio öffnen lässt. */ios* enthält dasselbe für Apple, während das *node_modules*-Verzeichnis die diversen JavaScript-Bibliotheken bereitstellt. Die beiden *index.*.js*-Dateien enthalten die eigentliche Applikationslogik, der wir uns im nächsten Schritt zuwenden wollen.

Zuerst soll das Android-Projekt in Android Studio geladen werden. Schließen Sie dazu ein eventuell geöffnetes Projekt durch Klick auf *File* und *Close Project* und klicken Sie dann im Sprungbildschirm auf die Option *Import project*. Wählen Sie die Datei *build.gradle* aus und klicken Sie auf *OK*, um den Ladeprozess anzustoßen. Aktuelle Versionen von Android Studio zeigen dabei mitunter die in Bild 1 gezeigte Fehlermeldung an. Das Anklicken von *Update* führt nicht zu Problemen.



Das Deployment von React-Native-Code erfolgt zweistufig (Bild 2)

An dieser Stelle müssen Android-Entwickler ein wenig umdenken. Die Initialisierung des Projekts erfolgt zwar nach wie vor über eine Activity, diese ist nun aber wie im folgenden Listing gezeigt aufgebaut:

```
public class MainActivity extends ReactActivity {
  @Override
  protected String getMainComponentName() {
    return "NMGTest1";
  }
  @Override
  protected boolean getUseDeveloperSupport() {
    return BuildConfig.DEBUG;
  }
  @Override
  protected List<ReactPackage> getPackages() {
    return Arrays.<ReactPackage>asList(
      new MainReactPackage()
    );
  }
}
```

ReactActivity dient als Host für die in der Einleitung beschriebene Laufzeitumgebung. Die Funktion *getUseDeveloperSupport* weist die Runtime darauf hin, dass das Programm im Moment im Debug-Betrieb ist. *GetPackages* ist für das Ermitteln einer Liste von verwendeten Paketen zuständig.

Da wir im Moment nur die in React Native enthaltenen Funktionen nutzen, wird das Array nur mit *MainReactPackage* bevölkert.

getMainComponentName dient als Schnittstelle zwischen nativem und JavaScript-Code. Sie weist *ReactActivity* darauf hin, dass der Hauptteil des Benutzerinterfaces in einer ▶

Komponente namens *NMGTest1* liegt. *index.android.js* beginnt mit der Inklusion einiger Module:

```
import React, { Component } from 'react';
import {
  AppRegistry,
  ...
} from 'react-native';
```

Die eigentliche Intelligenz unserer Applikation findet sich in der Klasse *NMGTest1*. React Native setzt das Vorhandensein von ES6 voraus, weshalb das Kommando *class* direkt eingebunden wird:

```
class NMGTest1 extends Component {
  render() {
    return (
      <View style={styles.container}>
        ...
        <Text style={styles.instructions}>
          Shake or press menu button for
          dev menu
        </Text>
      </View>
    );
  }
}
```

Facebook liefert das Projektskelett mit einem vergleichsweise komplexen Formular aus, dessen Struktur für uns allerdings nicht weiter interessant ist. Das einzige für uns relevante Detail ist, dass das anzuzeigende Markup in *return()* in Richtung des Frameworks geschickt wird.

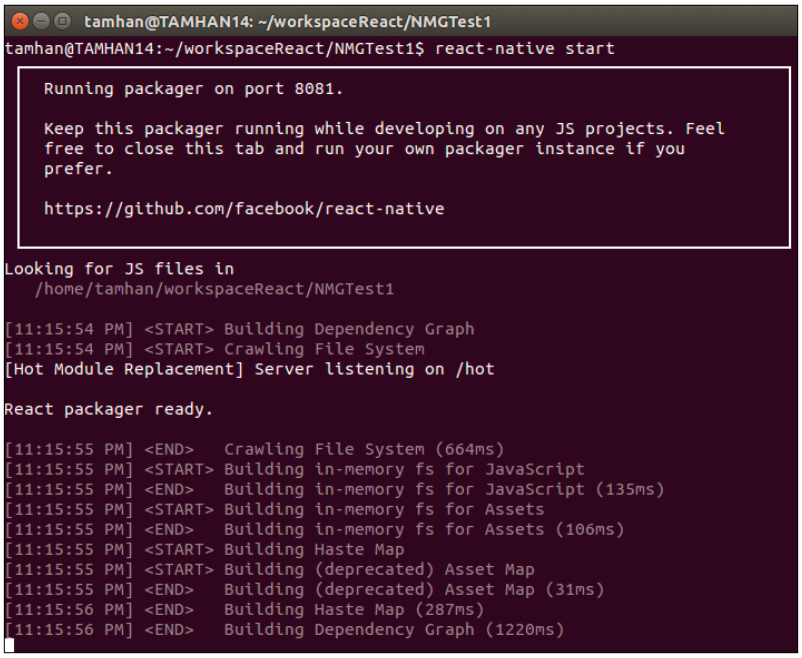
Arcunamagischer Transfer

Die eigentliche Registrierung der JavaScript-Klasse erfolgt in der letzten Zeile von *index.android.js*, wo die statische Methode *registerComponent* die Zuordnung zwischen String und Objekt herstellt:

```
AppRegistry.registerComponent('NMGTest1', () =>
NMGTest1);
```

Auch in Zeiten von USB 3.0 nimmt das Deployment von Programmen auf Smartphones viel Zeit in Anspruch. Die auf dem Papier harmlos erscheinende Auslieferungszeit von zwischen 10 und 30 Sekunden erweist sich in der Praxis als immenses Hindernis beim schnellen Testen von Programmänderungen.

React Native lässt sich bei der Lösung dieses Problems von der klassischen Webentwicklung inspirieren. Die auf dem Smartphone befindliche JavaScript-Runtime beschafft sich den auszuführenden Code separat von einem dedizierten Server. Insbesondere bei kleinen Änderungen ist das in Bild 2 gezeigte Verfahren einem vollständigen Depoyment überlegen.



React-native start zeigt sich kommunikativ (Bild 3)

Die Bereitstellung des HTML-Teils erfolgt durch einen in React Native enthaltenen Webserver, der sich im Projektverzeichnis durch Eingabe von *react-native start* starten lässt. Das Produkt zeigt sodann – wie in Bild 3 gezeigt – Statusinformationen über eingehende Anfragen an. Achten Sie darauf, dass das Terminal während des gesamten Entwicklungsprozesses geöffnet bleiben muss.

Verbinden Sie Ihr Telefon sodann mit der Workstation. Der Aufruf des im SDK befindlichen ADK-Kommandos sollte das Gerät erkennen:

```
tamhan@TAMHAN14:~/Android/Sdk/platform-tools$ ./adb
devices
List of devices attached
UCHQDQW499999999    device
```

Die Auslieferung lässt sich durch Eingabe von *react-native run-android* anwerfen. Im Rahmen des ersten Deployments lädt React Native einige Archive aus dem Internet herunter – unter anderem eine komplette Kopie des Gradle-Buildsystems.

In vielen Fällen kommt es im Rahmen der Auslieferung des Programms zu Fehlern, die sich durch Probleme im Bereich der Ausführung des Jobs *installDebug* ausdrücken. Schließen Sie in diesem Fall Android Studio und öffnen Sie die Datei *build.gradle*. Senken Sie die Version des zu verwendenden Plug-ins sodann auf Version 1.2.3, was durch folgende Anpassung zu bewerkstelligen ist:

```
buildscript {
  repositories {
    jcenter()
  }
  dependencies {
```

```
    classpath 'com.android.tools.build:gradle:1.2.3'
// NOTE: Do not place your application dependencies
// here; they belong in the individual module
// build.gradle files
  }
}
```

Insbesondere auf Telefonen mit 64-Bit-Prozessor und Android L funktioniert die Auslieferung über das Kommandozeilenwerkzeug oft eher schlecht als recht. Weitere Informationen dazu finden sich in der unter https://github.com/facebook/react-native/issues/2720 bereitgestellten Diskussion.

Der Autor löst dieses Problem durch Deployment aus Android Studio. Machen Sie hierzu die im vorigen Abschnitt beschriebene Änderung rückgängig und öffnen Sie das Projekt abermals in Googles IDE. Debuggen Sie es sodann wie eine gewöhnliche Android-Applikation. Dem nativen Teil des Projekts ist es völlig egal, wie er aufs Telefon kommt.

Achten Sie darauf, dass React Native bei Deployment-Fehlern oft nur einen weißen Bildschirm anzeigt. Weitere Informationen über die Fehlerursache finden sich in LogCat: Es zahlt sich bei seltsamem Verhalten immer aus, einen Blick auf das betreffende Fenster in Android Studio zu werfen (Bild 4).

In unserem Fall hakt die Auslieferung am Fehlen einer direkten TCP-Verbindung zwischen Telefon und Workstation. Ab Android 5.0 lässt sich dieses Problem durch die Eingabe des folgenden ADB-Kommandos lösen:

```
tamhan@TAMHAN14:~/Android/Sdk/platform-tools$ ./adb
reverse tcp:8081 tcp:8081
```

adb reverse weist ADB dazu an, den Port 8081 des Hostsystems für das gerade angeschlossene Telefon ansprechbar zu machen. Führen Sie sodann ein weiteres Deployment aus. Scheitert es immer noch mit einem Fehler der Bauart *Couldn't get the native call queue: bridge configuration isn't available*, so hilft normalerweise die Eingabe von *rm -rf node_modules && npm install*.

Nach getaner Arbeit – achten Sie darauf, dass der Codeserver läuft – präsentiert sich unsere Applikation wie in Bild 5 gezeigt.

Mach es in JavaScript

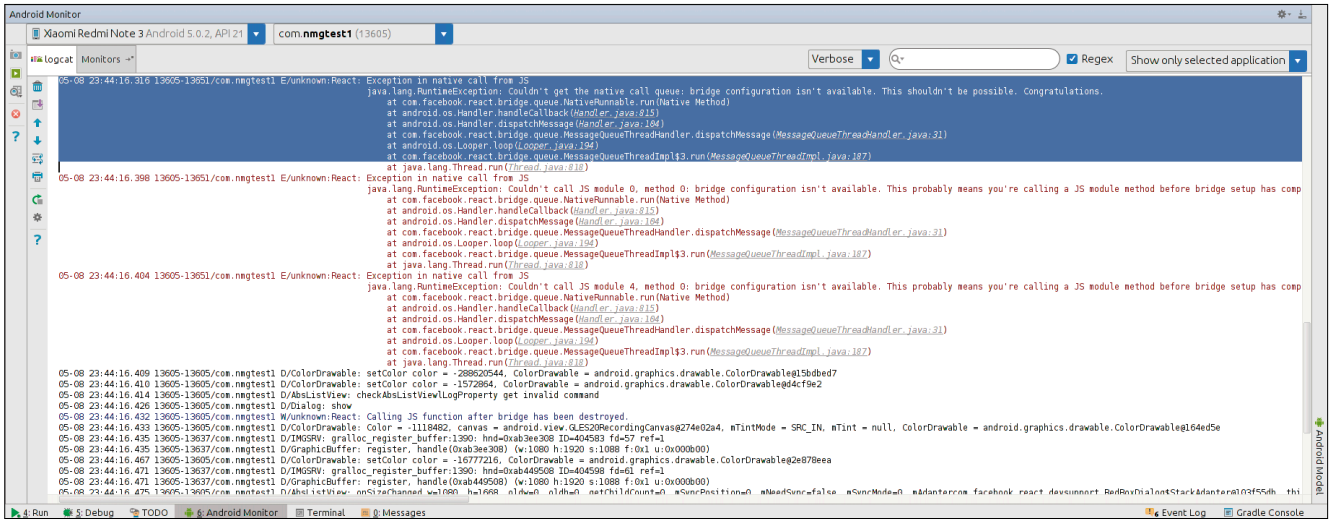
Frameworks wie React Native werden vor allem deshalb eingesetzt, um sich Arbeiten mit nativem Code zu ersparen. Wir wollen als kleines Beispiel eine App realisieren, die das Herunterladen von Informationen vom kanadischen Edelmetallhändler Kitco ermöglicht.

Als Erstes brauchen wir eine Datenbasis, die Informationen über die zu verarbeitenden Felder anliefert. Aus Gründen der Bequemlichkeit greifen wir hier auf ein statisches Array zurück, das im Anfang von *index.android.js* zu liegen kommt und folgendermaßen aussieht:

```
var EDELMETALL_DATEN = [
 {name: 'Gold', farbe: '2015', kitco:
 "http://www.kitco.com/images/live/gold.gif"},
 {name: 'Silber', farbe: '2015', kitco:
 "http://www.kitco.com/images/live/silver.gif"},
 {name: 'Platin', farbe: '2015', kitco:
 "http://www.kitco.com/images/live/plati.gif"},
];
```

Die Anzeige von kürzeren oder längeren Listen ist ein klassisches Performanceproblem, da die Generierung und Zerstörung der für die Einzelzellen zuständigen Widgets viel Rechenleistung in Anspruch nimmt. React Native begegnet diesem Problem durch die sehr leistungsfähige Klasse *ListView*, die unter https://facebook.github.io/react-native/docs/listview.html vollständig dokumentiert ist. Für ihre Nutzung muss die am Beginn der Datei stehende Import-Deklaration im ersten Schritt folgendermaßen angepasst werden:

```
import {
  AppRegistry,
  StyleSheet,
```



Während auf dem Smartphone nur ein weißer Bildschirm sichtbar ist, ist der Fehler in LogCat klar erkennbar (Bild 4)

```
  Text,
  ListView,
  View
} from 'react-native';
```

ListViews arbeiten auf Basis von Zuständen. Unsere Klasse *NMGTest1* hat im Moment keinen Konstruktor und weist deshalb keinen Initialzustand auf. Dank ECMAScript 6 können wir hier direkt auf das *constructor*-Schlüsselwort zurückgreifen:

```
class NMGTest1 extends Component {
  constructor(props) {
    super(props);
    var ds = new ListView.DataSource({rowHasChanged:
    (r1, r2) => r1 !== r2});
    this.state= {dataSource: ds.cloneWithRows
    (EDELMETALL_DATEN)};
  }
```

An dieser Routine sind zwei Aspekte interessant. Erstens beziehen ListViews ihre Elemente immer aus Data-Source-Elementen, die für die Erkennung der Gleichheit oder Ungleichheit zweier Reihen eine Komparatorfunktion mitbringen.

Zweitens fragen sich aufmerksame Leser an dieser Stelle, warum wir das Setzen des Status nicht einfach in Render ermöglichen. Das liegt daran, dass Zustandsänderungen im Rahmen des Renderings nicht erlaubt sind und während der Programmausführung zu diversen, nur in LogCat erkennbaren Fehlern führen.

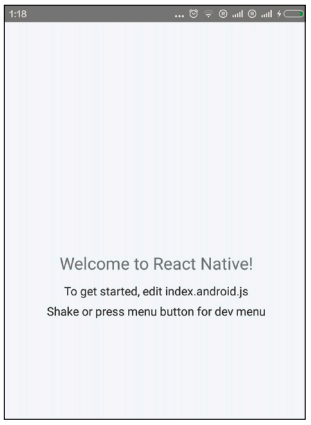
Als Nächstes wenden wir uns der Renderingfunktion zu, die die ListView in den Bildschirm schreibt. Neben dem Wurzel-Tag setzen wir hier auch einige Attribute, die das Verhalten der Liste zur Laufzeit beeinflussen:

```
render(){
  return (<ListView
    dataSource={this.state.dataSource}
    renderRow={this.renderMetall}
    style={styles.listView} /> );
}
```

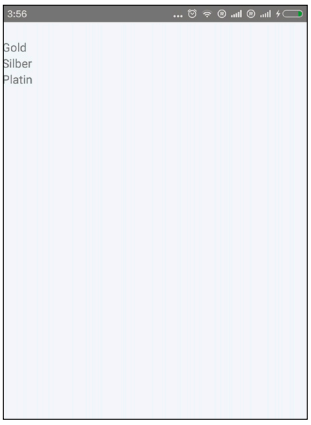
ListViews stellen die ihnen zugewiesenen Elemente über eine Gruppe von Untersteuerelementen dar, die im Rahmen einer *RenderRow*-Funktion erzeugt werden. Im Fall unseres Beispiels heißt diese Methode *renderMetall:*

```
 renderMetall(metal){
  console.log("rendermetall aktiv");
    return (
      <Text>{metal.name}</Text>
    );
  }
}
```

Hier ist Zweierlei interessant: Neben dem Zurückgeben eines per Data Binding aufgebauten Steuerelements rufen wir die Methode *console.log* auf, die eine Meldung in das LogCat-Fenster der IDE schreibt.



Das Projektskelett auf dem Smartphone (Bild 5)



Die drei Edelmetalle erscheinen am Bildschirm (Bild 6)

Für die Darstellung der ListView ist eine kleine Änderung am Stylesheet notwendig, das nun folgendermaßen aussieht:

```
const styles = StyleSheet.create({
  container: {
    flex: 1,
    justifyContent: 'center',
    alignItems: 'center',
    backgroundColor: '#F5FCFF',
  },
  listView: { paddingTop: 20,
  backgroundColor: '#F5FCFF', },
});
```

Damit ist unser Programm auch schon zur Ausführung bereit. Die in der Liste enthaltenen Namensinformationen scheinen wie in Bild 6 gezeigt am Bildschirm auf. Farben und URL-Daten kommen im Moment noch nicht zum Einsatz.

Aktualisiere automatisch

Änderungen im JavaScript-Teil erfolgen normalerweise außerhalb von Android Studio. Auch wenn der React-Native-Server Modifikationen theoretisch automatisch erkennen sollte, funktioniert dies in der Praxis eher schlecht als recht.

Aufgrund von Caching sollten Sie die Applikation vor dem Deployment komplett vom Telefon löschen. Wer dies nicht von Hand bewerkstelligen möchte, kann die Aufgabe auch an das Gradle-Buildsystem delegieren. Klicken Sie dazu auf *Run* und *Edit Configurations* und wechseln Sie in den Tab *General*. Klicken Sie danach auf das Plus-Symbol in der Rubrik *Before Launch*. Im Kontextmenü folgt die Auswahl von *Gradle-Aware Make*. Geben Sie im daraufhin erscheinenden Feld den Befehl *:app:uninstallDebug* ein und verschieben Sie ihn wie in Bild 7 gezeigt an die Spitze des Bauprozesses.

Leider eliminiert diese Vorgehensweise die Vorteile der im vorigen Abschnitt besprochenen Serverarchitektur. Zudem ist eine Komplettaktualisierung unnötig. Es wäre wesentlich effizienter, wenn der geänderte Code im Stil von NativeScript

automatisch heruntergeladen werden würde. Dies lässt sich über das Developer-Menü bewerkstelligen, das durch Drücken der Menütaste auf den Bildschirm geholt wird und die in Bild 8 gezeigten Auswahlfunktionen anbietet. Die Funktion *Reload JS* weist das Programm dazu an, den auf der lokalen Maschine arbeitenden React-Server um eine neue Version des JavaScripts zu erleichtern.

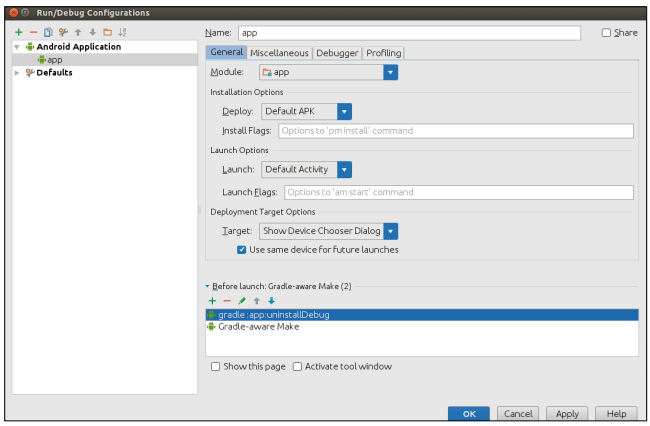
Linux bietet nur wenige Möglichkeiten zur Überwachung der Aktualität einzelner Dateien an. Facebook behebt dieses Problem mit einem Produkt namens Watchman. Wenn Sie Probleme mit der Auslieferung veralteter Dateiversionen haben und die im Installationsabschnitt beschriebenen Kommandos das Problem nicht lösen, sollten Sie das Programm folgendermaßen kompilieren:

```
git clone https://github.com/facebook/watchman.git
cd watchman
git checkout v4.5.0
./autogen.sh
./configure
make
sudo make install
```

Auch wenn die App mittlerweile funktioniert: Besonders gut sieht das gerenderte Resultat nicht aus. Als ersten Schritt zur Verbesserung des Aussehens der generierten Tabelle ist eine kleine Änderung im Edelmetall-Daten-Array notwendig: Sstatt der bisher verwendeten Zahlen setzen wir nun HTML-Farbkonstanten ein:

```
var EDELMETALL_DATEN = [
  {name: 'Gold', farbe: '#FFCC66', ...
  {name: 'Silber', farbe: '#CCCCCC', ...
  {name: 'Platin', farbe: '#AAAAAA', ...
];
```

Im nächsten Schritt müssen wir die Struktur der zurückgelieferten Items anpassen. Für die Anzeige der Farbe des jeweiligen Metalls verwenden wir eine View, deren *BackgroundColor*-Attribut auf den gewünschten Wert gesetzt wird.



Android Studio kann die Applikation vor jedem Deployment deinstallieren (Bild 7)

Um die beiden Steuerelemente nebeneinander anzuordnen, nutzen wir eine weitere View. Das Endresultat ist also eine Containerview, die zwei nebeneinanderliegende Steuerelemente enthält. Dazu ist in *renderMetall* folgende Änderung notwendig:

```
renderMetall(metal){
  console.log("rendermetall aktiv");
    return (
      <View style={styles.elemContainer}>
        <View style={styles.colorContainer}
        backgroundColor={metal.farbe}></View>
        <Text>{metal.name}</Text>
      </View>
    );
  }
}
```

Ein Gutteil der Festlegung des Aussehens erfolgt in React Native durch CSS-artige Styles. In unserem Programmbeispiel beziehen wir alle Styles aus dem im Rahmen der Projekterstellung erzeugten StyleSheet-Objekt, das nun um die folgenden Attribute erweitert wird:

```
const styles = StyleSheet.create({
  ...
  elemContainer:{
    flex: 1,
    flexDirection: 'row',
    justifyContent: 'center',
    alignItems: 'center',
    backgroundColor: '#F5FCFF',
  },
  colorContainer:{
    width: 80,
    height: 80,
  }, });
```

React Native nutzt für das Ausrichten der Steuerelemente am Bildschirm die vom W3C mittlerweile fast komplett durchstandardisierte Flexbox-Komponente: Ähnlichkeiten zu den aus diversen Cross-Plattform-Entwicklungssystemen bekannten Layoutsystemen sind rein zufällig.

Wir erzeugen hier zwei Steuerelemente: *elemContainer* ist durch Setzung von *flex=1* ein dem Layoutprozess unterliegendes Widget, das seine Kinder getreu der *FlexDirection*-Eigenschaft in Reihen anordnet.

Finalisieren des Aussehens

colorContainer ist ob der fixen Werte für *width* und *height* nicht layoutbar. Seine Größe ist von den Systemumgebungsbedingungen unabhängig. Wer das Programm nun ausführt, bekommt das in Bild 9 gezeigte Resultat. Das liegt daran, dass die Textbox sich nicht maximal ausbreitet.

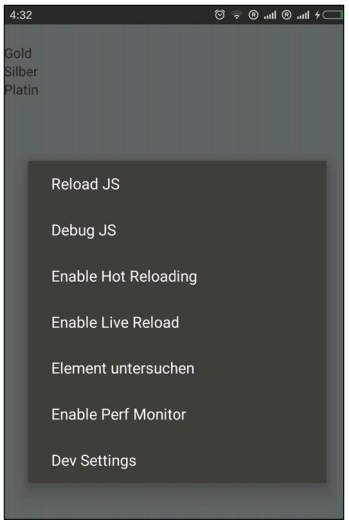
Dieses Problem tritt auf, weil die Textbox im Moment kein Teil des Flexlayouts ist und somit nicht an Breite zunimmt. Als ersten Schritt zur Finalisierung des Aussehens passen wir ▶

renderMetall abermals an, um der Textbox nun den Stil *textContainer* zuzuweisen:

```
renderMetall(metal){
...
  return (
    ...
    <Text style={styles.textContainer}>
    {metal.name}</Text>
</View>
); } }
```

Im Stylesheet legen wir eine weitere Style-Deklaration an, die sich auf das Setzen der Flex-Eigenschaft beschränkt:

```
const styles = StyleSheet.create({
  ...
  textContainer: {
    flex: 1, },
});
```



Das Entwicklermenü bietet diverse Optionen an (Bild 8)

Damit sind wir zur Programmausführung bereit. Bild 10 zeigt das Resultat unserer Bemühungen. Das korrekte Verhalten tritt auf, weil die Textbox durch das Setzen von Flex nun ebenfalls den Layoutrichtlinien der FlexBox unterliegt und somit maximal ausgebreitet wird. Dadurch wird der farbige Kasten an den linken Rand des Bildschirms geschoben.

Wechsle die Seiten

Damit haben wir die Anzeige einer Liste von Edelmetallen bewerkstelligt. Als nächste Aufgabe präsentiert sich das Anzeigen der von Kitco seit Jahren unter ein und demselben URL bereitgestellten Diagramme. Dazu müssen wir die Struktur unseres Programms umstellen: Anstelle der direkt rendernden Single-Page-Applikation müssen wir nun ein Navigator-Steuerelement erzeugen. Falls Sie dieses Tutorial Schritt für Schritt nachvollziehen, sollten Sie an dieser Stelle eine Sicherheitskopie des Codeordners anlegen.

Öffnen Sie danach die Datei *index.android.js* und ersetzen Sie das in der *render*-Funktion befindliche Markup durch folgende Deklaration eines Navigator-Steuerelements:

```
class NMGTest1 extends Component {
render(){
  return ( <Navigator
    initialRoute={{id: 'ListPage', name: 'Index'}}
    renderScene={this.renderScene.bind(this)}
    configureScene={(route) => {
    if (route.sceneConfig) {
      return route.sceneConfig;
    }
    return Navigator.SceneConfigs.VerticalDownSwipeJump;
}}/>); }
```

Navigator-Steuerelemente agieren als eine Art Makler zwischen verschiedenen Containerklassen. *initialRoute* legt dabei die nach der Erstellung der Komponente anzuzeigende Seite fest. In der im folgenden Schritt besprochenen Funktion *renderScene* findet sich die eigentliche Logik zum Wechseln der aktiven Inhalte, während *configureScene* für die Bestimmung der zu verwendenden Animation zuständig ist.

RenderScene wird mit dem Namen der Route und einem Verweis auf das für die Routenänderung zuständige Navigator-Objekt aufgerufen. Zweiteres stellt eine wichtige Kontextinformation dar, die unter anderem zum Parametrieren der zu öffnenden Seiten herangezogen werden kann:

```
renderScene ( route, navigator ) {
  var routeId = route.id;
  if (routeId === 'ListPage') {
    return (<ListPage
navigator={navigator}/>);
  }
  if (routeId === 'ChartPage') {
    return (<ChartPage navigator={navigator}/>);
} }
```

Zu guter Letzt muss die Importdeklaration in der Indexdatei um einen Verweis auf die Navigatorklasse ergänzt werden. Unterbleibt dies, so würde der am Telefon befindliche Runner die Programmausführung mit einem roten Fehlerbildschirm unterbrechen:

```
import {
  ...
  View,
  Navigator
} from 'react-native';
```

Als nächste Aufgabe müssen wir die beiden Tags *ListPage* und *ChartPage* beleben. Dazu erzeugen wir zwei neue Dateien namens *listpage.js* und *chartpage.js*, die auf einer Ebene mit *package.js* zu liegen kommen. Die bei den Indexdateien verwendete Qualifikation durch Einfügen von *.android* oder *.ios* ist für uns an dieser Stelle insofern uninteressant, als dieses Beispiel nur für Android vorgesehen ist.

In *ListPage* entsteht eine neue Klasse, die den bisher zur Realisierung der Liste verwendeten Code aufnimmt. Da die Implementierungen der Methoden keinen Unterschied aufweisen, drucken wir im Listing nur die Outline der Klassenstruktur ab:

```
import React, { Component } from 'react';
...
export class ListPage extends Component {
  constructor(props) {
   ...
   render(){
     return (<ListView
```

```
    ...
    renderMetall(metal){
    ...
  }
  const styles = StyleSheet.create({
  ...
}
```

ChartPage ist im Moment noch nicht implementiert. Zur Erleichterung eines Funktionstests des Navigators errichten wir in der Datei einen Klassenstumpf, der neben der obligatorischen Exportierung auch eine Lebensäußerung emittiert:

```
import React, { Component } from 'react';
...
export class ChartPage extends Component {
  render(){
    return (<Text>"Hallo Welt"</Text>);
} }
```

Einzelne Views werden bei React Native nicht unter Nutzung der *AppRegistry*-Klasse angemeldet. Sie ist nur dann notwendig, wenn das betreffende Element aus der nativen Umgebung heraus angesprochen werden muss und – wie die Page mit dem Navigator – als Einsprungpunkt dient.

Die einzelnen Seiten sind dank ECMAScript 6 als Module aufgebaut, die unter Nutzung des *export*-Befehls je eine Klasse exponieren. Daraus folgt, dass die Einbindung wie bei Framework-Elementen erfolgt – in der Indexdatei ist folgende Erweiterung notwendig:

```
import {ListPage} from './listpage.js';
import {ChartPage} from './chartpage.js';
```

Damit ist unsere Applikation abermals zur Ausführung bereit. Aus optischer Sicht ergibt sich im Moment kein wesentlicher Unterschied, da der Navigator ja nur die schon bekannte View lädt.

Unsere ListView ist im Moment nicht zur Entgegennahme von Tipp-Events befähigt. React Native bietet mit *TouchableNativeFeedback* eine darauf spezialisierte Klasse an, die zudem für die notwendige Aktualisierung des Aussehens der betreffenden Steuerelemente sorgt. Ihre Einbindung setzt eine kleine Änderung in *renderMetall* voraus:

```
renderMetall(metal){
  return (  <TouchableNativeFeedback onPress=
    {this.onPressButton} background=
    {TouchableNativeFeedback.SelectableBackground()}>
      <View style={styles.elemContainer}>
        <View style={styles.colorContainer}
        backgroundColor={metal.farbe}></View>
        <Text style={styles.textContainer}>{metal.name}
        </Text>
      </View>
  </TouchableNativeFeedback>);
}
```

Das Nicht-Entfernen der inneren View ist notwendig, weil *TouchableNativeFeedback*-Instanzen nur ein Kindelement aufnehmen dürfen. Wer diese Regel verletzt, riskiert undefiniertes und potenziell fehlerhaftes Verhalten.

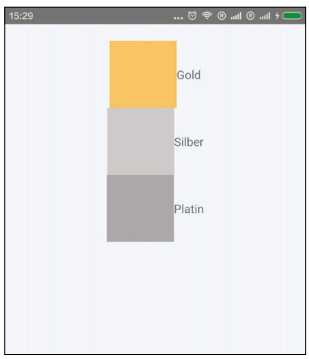
Da wir *TouchableNativeFeedback* im Moment nur in *listpage.js* verwenden, ist die Anpassung der *import*-Deklaration nur in ebendieser Datei erforderlich:

```
import {
  ...
  View,
  TouchableNativeFeedback
} from 'react-native';
```

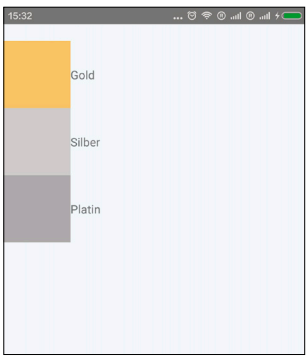
Wenden wir uns nun der Implementierung der Routine zu, die die eigentlichen Klicks entgegennimmt. Ein erster, naiver Aufbau würde *onPressButton* in *ListPage* implementieren:

```
export class ListPage extends Component {
  ...
  onPressButton(event) {
    console.log('Pressed!');
}
```

Wer diese Version des Programms aufs Telefon schickt und auf das Listenelement klickt, sieht außer der Veränderungsanimation nichts – auch in LogCat bleibt die Lage ruhig. Daraus folgt, dass die Methode offensichtlich nicht abgearbeitet wird – die relevante Frage lautet: Warum?



Die Edelmetalle schweben in der Mitte der Liste (Bild 9)



Unsere Liste ist einsatzbereit (Bild 10)

Eine Frage des Kontexts

Erfahrene Verfechter von sauberem JavaScript warnen bisweilen davor, allzu sehr auf *this* zu vertrauen. Der Wert der Variable kann oft überraschend sein. Dies trifft auch für unser Beispiel zu. *this* zeigt beim Aufruf des Event Handlers mit hoher Wahrscheinlichkeit auf ein Steuerelement.

Da das Anlegen einer globalen Variable der Codequalität nicht förderlich ist, greift man an dieser Stelle am besten auf das Konzept der Closures zurück. Es handelt sich dabei um eine kleine Besonderheit von JavaScript, die in Bild 11 illustriert ist. ▶

Mit einer Closure vereinfacht sich das Einschreiben eines Event Handlers massiv. Die Variable *onPressButton* bleibt auch nach der Abarbeitung von *renderMetall* enthalten und stellt den in ihr angelegten Event Handler so dem Steuerelement zur Verfügung:

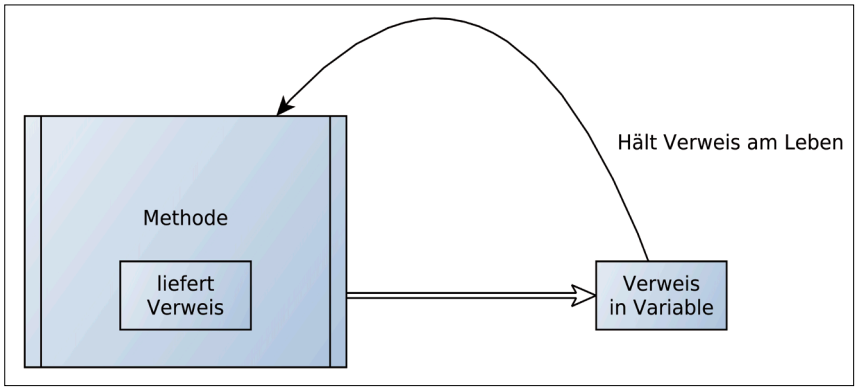
```
renderMetall(metal){
  var onPressButton=function()
  {
    console.log("Hallo!");
  };
  return (  <TouchableNativeFeedback onPress=
  {onPressButton} background=
  {TouchableNativeFeedback.SelectableBackground()}>
    ...
    </TouchableNativeFeedback>);
}
```

Wenn Sie das vorliegende Programm auf Ihrem Smartphone ausführen und die Liste kräftig anklicken, so passiert – neben der schon im vorigen Abschnitt gesehenen Animation – nichts. Kurze Klicks führen hingegen zur Ausgabe der Meldung in LogCat.

Die Ergründung dieses Verhaltens wird durch den optischen Aufbau der Dokumentation von React Native erschwert. Wer die unter https://facebook.github.io/react-native/docs/touchablenativefeedback.html bereitgestellte Hilfeseite des Steuerelements aufruft, ist mit dem in Bild 12 gezeigten Resultat konfrontiert.

Facebook druckt die ererbten Eigenschaften von Steuerelementen und Klassen nicht separat ab. Der einzige Verweis auf ihr Vorhandensein ist der blaue Link in der Rubrik *props*. Wer ihn anklickt, landet in den Eigenschaften des Mater-Elements. Dort weist man darauf hin, dass *longPresses* das *onPress*-Event kapern können: Wer sich an langsamere User wenden möchte, baut *renderMetall* folgendermaßen auf:

```
renderMetall(metal){
   var onPressButton=function() {
     ...
   return (  <TouchableNativeFeedback onPress=
   {onPressButton} onLongPress={onPressButton}  ...
```



Closures verhindern die automatische Abtragung von Speicherbereichen (Bild 11)

Damit bleibt nur noch eine Aufabe ungelöst: Die Diagrammdaten müssen vom Server von Kitco heruntergeladen und im Formular angezeigt werden.

RenderMetall() läuft zur Gänze außerhalb des *this*-Kontexts des Host-Steuerelements ab, weshalb der Zugriff auf das Navigatorobjekt etwas Denkarbeit voraussetzt. Der einfachste Weg zur Lösung des Problems besteht darin, den *this*-Kontext von *renderMetall* zu bestimmen. In ECMAScript6-Klassen erfolgt dies durch folgende kleine Änderung im Konstruktor:

```
export class ListPage extends Component {
 constructor(props) {
  ...
  this.renderMetall = this.renderMetall.bind(this);
}
```

Ab diesem Zeitpunkt lässt sich die Navigator-Instanz nach folgendem Schema anrufen:

```
renderMetall(metal){
  var nav=this.props.navigator;
    var onPressButton=function()
    {
      nav.replace({id: 'ChartPage'});
    };
```

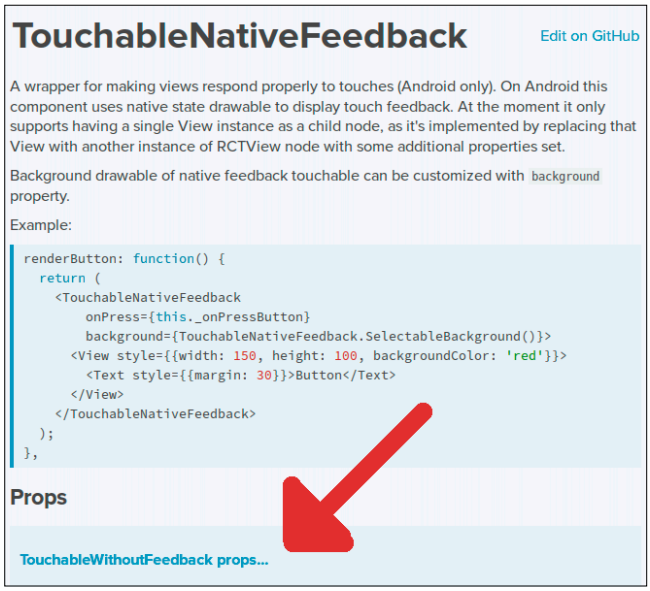
Im nächsten Schritt müssen wir *ChartPage* zur Anzeige des Diagramms animieren. Im Konstruktor schreiben wir den URL des Diagramms für den Silberkurs ein, um die View auf das Herunterladen von Bilddateien zu parametrieren:

```
export class ChartPage extends Component {
  constructor(props) {
    super(props);
    this.state = {imageURL: "http://www.kitco.com/
    images/live/silver.gif"};
    this.chartpageupdater=this.chartpageupdater.
    bind(this);
    props.navigator.chartpageupdater=
    this.chartpageupdater;
  }
```

Für die eigentliche Zustandsänderung ist *chartpageupdater* verantwortlich. Um anderen Komponenten den Zugriff auf die Methode zu ermöglichen, wird sie im Rahmen des Konstruktors in das Navigator-Objekt geschrieben:

```
chartpageupdater(url){
  this.setState({imageURL: url});
  console.log(url);
}
```

Das eigentliche Erzeugen des für die Anzeige des Charts notwendigen DOM-



Der vom Autor eingezeichnete rote Pfeil zeigt, wo wichtige Informationen warten (Bild 12)

Baums erfolgt im Rahmen der *render()*-Methode der betreffenden Komponente. Für die Anzeige eines Bilds ist ein *Image*-Steuerelement erforderlich:

```
render(){
  return (<View style={chartstyles.container}>
    <Image source={{uri:this.state.imageURL}}
    style={chartstyles.thumbnail}/>
  </View>);
}}
```

Für die Darstellung ist zudem ein grundlegendes Stylesheet notwendig, das wir lokal in der Datei *chartpage.js* anlegen:

```
var chartstyles = StyleSheet.create({
  container: { flex: 1,
    justifyContent: 'center',
    alignItems: 'center',
    backgroundColor: '#F5FCFF', },
  thumbnail: {width: 300, height: 200, }, });
```

Damit fehlt nur noch der Aufruf von *chartpageupdater*, der im Rahmen von *onPressButton* nach folgendem Schema erfolgt:

```
renderMetall(metal){
  var url=metal.kitco;
  var nav=this.props.navigator;
  var onPressButton=function() {
    nav.replace({id: 'ChartPage'});
    nav.chartpageupdater(url);
  };
```

Führen Sie nun das Programm aus und klicken Sie auf einen der Einträge. Das betreffende Chart erscheint am Bildschirm. React Native handhabt den *Back*-Button nicht selbst: Zur Implementierung bietet sich die Nutzung der *BackAndroid*-Klasse an.

Programm auslieferbereit machen

React Natives serverbasierte Architektur ist für die Entwicklung von Programmen hilfreich – Kunden brauchen so den auszuführenden Code nicht jedes Mal neu herunterzuladen.

Erfreulicherweise bringt das vom React Native CLI erzeugte Projektskelett die für die Kompilation eines Release-APK notwendigen Werkzeuge mit. Die Erzeugung des APK lässt sich im Projektverzeichnis durch die Eingabe der folgenden beiden Befehle bewerkstelligen:

```
cd android && ./gradlew assembleRelease
```

Zur Abarbeitung dieses Teils der Toolchain sind einige Kommandozeilenvariablen erforderlich, die auf den zur Signierung zu verwendenden Keystore hinweisen. Facebook beschreibt die erforderliche Konfiguration im Detail unter https://facebook.github.io/react-native/docs/signed-apk-android.html. Sind die betreffenden Variablen nicht gesetzt, so beginnt React Native stattdessen mit einem normalen Kompilationsprozess, der je nach Systemgeschwindigkeit bis zu fünf Minuten in Anspruch nimmt:

```
tamhan@TAMHAN14:~/workspaceReact/NMGTest1$ cd android/
tamhan@TAMHAN14:~/workspaceReact/NMGTest1/android$ ./
gradlew assembleRelease
Incremental java compilation is an incubating feature.
...
BUILD SUCCESSFUL
Total time: 1 mins 27.525 secs
This build could be faster, please consider using the
Gradle Daemon: https://docs.gradle.org/2.10/userguide/
gradle_daemon.html
```

Die dabei entstehenden *.apk*-Dateien finden sich sodann in einem Unterverzeichnis des Android-Ordners.

Fazit

React Native wird in einer Vielzahl von Facebook-Projekten eingesetzt. Die Freigabe des Quellcodes erfolgte primär zur Ausnutzung des Linus-Effekts: Ist der Code eines Produkts für jedermann einsehbar, so werden Fehler schneller gefunden. Facebook profitiert zudem von der Verfügbarkeit fertig trainierter Entwickler. ■

Tam Hanna
ist Autor, Trainer und Berater mit den Schwerpunkten Webentwicklung und Webtechnologien. Er lebt in der Slowakei und leitet dort die Firma Tamoggemon Holding k.s. Er bloggt sporadisch unter:
www.tamoggemon.com

IOS AIRPRINT

Druck und Dokument

So übertragen Sie unter iOS Dokumente via WLAN an angeschlossene Drucker.

Ausdruck und Bearbeitung von Dokumenten scheint auf den ersten Blick nicht die typische Domäne von Smartphones oder Tablets zu sein, aber dem ist natürlich nicht so. Gerade das iPhone und vor allem das iPad werden in immer mehr Bereichen von Unternehmen eingesetzt. Gerade dort ist die Möglichkeit, einen Ausdruck oder ein Dokument zu erzeugen, sehr wichtig.

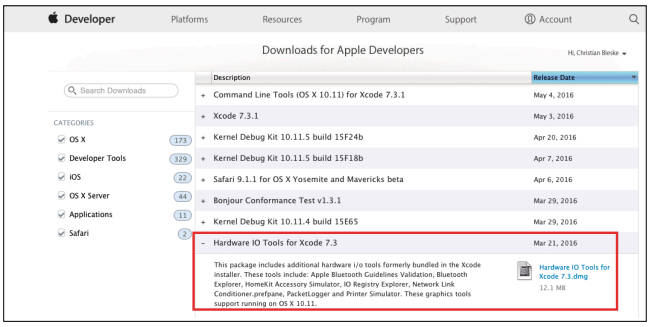
Aber auch im privaten Bereich gibt es sicherlich genügend Material zum Ausdruck – sei es, dass ein Brief ausgedruckt werden soll oder die Fotos aus dem letzten Urlaub. Nun bietet ein iPad oder auch ein iPhone keine direkte Möglichkeit zum Anschluss eines Druckers. Moderne Drucker aber lassen sich in der Regel über eine WLAN-Schnittstelle ansprechen. Auch Apple unterstützt diese Form der Kommunikation von iOS-Geräten mit Drucker unter der Bezeichnung AirPrint.

AirPrint ist eine Technologie von Apple, die es ermöglicht, Dokumente von einem iOS-Gerät aus an den Drucker zu schicken. Das Besondere an AirPrint ist, dass für eine Verbindung mit einem iOS-Gerät keine Druckertreiber auf dem iOS-Gerät installiert werden müssen. Für Entwickler hat Apple in iOS mehrere APIs integriert, die den einfachen Zugang zu dieser Schnittstelle aus Apps heraus ermöglichen. Bevor man sich einen Drucker zum Testen kauft, sollte man allerdings prüfen, ob dieser zu AirPrint kompatibel ist. AirPrint wird neben iOS auch von OS X unterstützt.

Druck ohne Drucker

Aber auch wenn man keinen Drucker zur Hand hat, muss man nicht darauf verzichten, die eigene App mit einer Druckunterstützung auszustatten. Testen lässt sich der Ausdruck auch mit dem sogenannten Printer Simulator. Es handelt sich dabei um eine Erweiterung für Xcode, die sich separat herunterladen lässt (Bild 1).

Nach dem Download öffnet ein Doppelklick die DMG-Datei. Via Drag and Drop kann unter anderem der Printer Simulator in ein zuvor angelegtes Verzeichnis kopiert werden. Der



Download der Hardware IO Tools for Xcode (Bild 1)

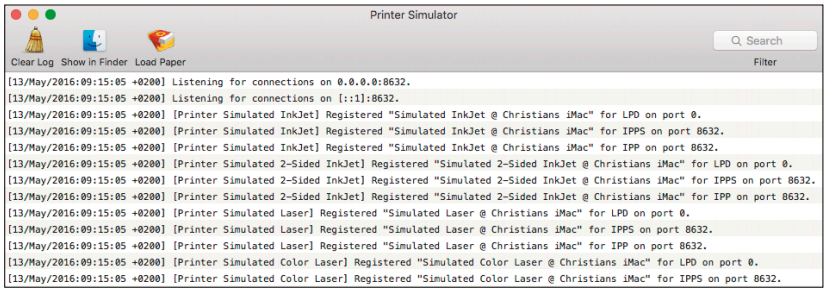
Printer Simulator muss nun nur noch durch einen Doppelklick gestartet werden. Anschließend läuft das Programm im Hintergrund und wartet darauf, dass aus dem iOS-Simulator heraus die Druckerschnittstelle angesprochen wird. Sobald das der Fall ist und ein Ausdruck in Auftrag gegeben wurde, wird der Printer Simulator aktiv. Er nimmt den Druckauftrag entgegen und leitet diesen um (Bild 2). Ein physischer Ausdruck wird natürlich nicht erstellt. Aber das Ergebnis kann innerhalb einer automatisch erzeugten PDF-Datei, die im Vorschau-Programm von OS X angezeigt wird, begutachtet werden.

Druck aus TextView

Um etwas ausdrucken zu können, muss die App im ersten Schritt das entsprechende API von iOS verwenden. Das geht rech simpel – wie Listing 1 demonstriert. In ein Single-View-Application-Projekt werden ein TextView-Control sowie ein Button in der View eingefügt. Anschließend werden ein Outlet für die TextView und eine Action für den Button erzeugt.

Nach beendeter Konfiguration der View sollte das Ergebnis wie in Bild 3 aussehen. Sobald der Button mit der Beschriftung *Ausdrucken* betätigt wurde, wird zuerst eine Instanz der Klasse *UIPrintInteractionController* angelegt. Diese Instanz wird benötigt, um den Ausdruck (PrintJob) später anstoßen zu können. Hierzu sind zuvor aber zwei Informationen erforderlich: der Name des Jobs und eine Instanz der Klasse *UIMarkupTextPrintFormatter*. Der Name wird der Instanz *uiPrintInfo* übergeben. Außerdem wird der Ausgabetyp *(OutputType)* festgelegt.

Dieser legt fest, ob Text, ein Foto oder ein Dokument mit Graustufen ausgedruckt werden soll. Diese Einstellung kann das Druckergebnis erheblich beeinflussen. Im Beispiel



Der Printer-Simulator im Einsatz (Bild 2)

Listing 1: Ausdruck im Code anstoßen

```
import UIKit

class ViewController: UIViewController {
  @IBOutlet var uiTextView: UITextView!
  override func viewDidLoad() {
    super.viewDidLoad()
  }
  override func didReceiveMemoryWarning()
  {
    super.didReceiveMemoryWarning()
  }

  @IBAction func button_Pressed(sender: AnyObject)
  {
  let uiPrintInteractionController =
    UIPrintInteractionController.
      sharedPrintController()
  let uiPrintInfo = UIPrintInfo(
    dictionary:nil)
  uiPrintInfo.outputType =
    UIPrintInfoOutputType.General
  uiPrintInfo.jobName = "Ausdruck"
  uiPrintInteractionController.printInfo =
  uiPrintInfo
  let uiMarkupTextPrintFormatter =
    UIMarkupTextPrintFormatter(
      markupText: uiTextView.text)
  uiMarkupTextPrintFormatter.contentInsets =
  UIEdgeInsets(top: 72,
               left: 72,
               bottom: 72,
               right: 72)
  uiPrintInteractionController.printFormatter =
    uiMarkupTextPrintFormatter
  uiPrintInteractionController.
    presentAnimated(true, completionHandler: nil)
  }
}
```

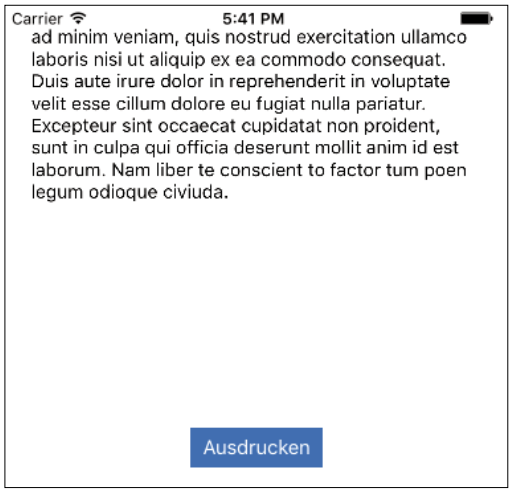
wurde die Einstellung *General* gewählt. Diese Auswahl legt fest, dass das Dokument einen Mix aus Text und Grafiken enthält. Die gewählte Konfiguration wird zuletzt der *UIPrintInteractionController*-Instanz zugewiesen.

Im letzten Schritt vor dem Ausdruck wird mittels einer Instanz der Klasse *UIMarkupTextPrintFormatter* festgelegt, welche Abmessungen das Dokument verwendet. Gestartet wird der Ausdruck durch einen Aufruf der Methode *presentAnimated* der Klasse *UIPrintInteractionController*. Zur Laufzeit der App wird in der App nach Betätigung des Buttons der Dialog *Printer Options* angezeigt (Bild 4). Im Dialog lassen sich unter anderem die Anzahl der Kopien sowie der verwendete Drucker einstellen. Ein Klick auf den *Print*-Button sorgt für den (physikalischen) Ausdruck.

Generierung eines PDF-Dokuments

Im vorangegangenen Beispiel wurde vorgestellt, wie sich der Ausdruck von Texten, die beispielsweise in einem TextView-Control vorhanden sind, bewerkstelligen lässt. Das ist zwar schon ganz nützlich, allerdings kommt es sicherlich auch hin und wieder vor, dass nicht nur der Inhalt eines Controls, sondern ein ganzes Dokument (inklusive Layout) ausgedruckt werden muss.

In diesen Fällen genügt es nicht, nur den Weg des Ausdrucks zu konfigurieren, sondern das komplette Dokument selbst muss konfiguriert beziehungsweise dessen Layout erstellt werden. In solchen Fällen ist es besser, dass man im Code ein PDF-Dokument erzeugt, das im nächsten Schritt ausgedruckt werden kann. Das PDF-Format ist aber nur der halbe Weg zum Ziel.



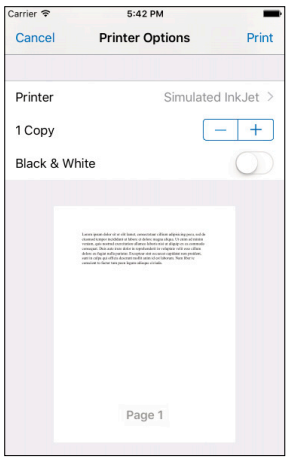
GUI des Testprogramms zum Ausdruck (Bild 3)

Um einen Text wie gewünscht formatieren zu können, benötigt man in der Regel eine Textverarbeitung oder zumindest eine entsprechende Auszeichnungssprache. Wie gut, dass mit HTML eine solche verfügbar ist, die problemlos in einem iOS-Programm verwendet werden kann. Listing 2 können Sie entnehmen, wie man einen Text mit HTML aufbereitet und anschließend ein PDF-Dokument daraus erzeugt.

Zuerst wird mit den Anlegen des Textes begonnen. Dieser wird zur einfacheren Formatierung zu Beginn in zwei Variablen gespeichert. Der Titel wird mittels der Auszeichnung *h1* entsprechend formatiert. Der restliche Text in der zweiten Variablen *htmlText* wird mittels der Auszeichnung *b* fett formatiert. Das soll für das Beispiel genügen. In den folgenden beiden Zeilen wird der Text in der Variablen *completeText* zusammengeführt. Anschließend geht es mit der Vorbereitung zur Generierung des PDF-Dokuments weiter.

PDF im Code erzeugen

Der erste Schritt zum PDF-Dokument ist die Ableitung einer Instanz der Klasse *UIMarkupTextPrintFormatter*. Mit dieser Klasse werden Texte im HTML-Format zum Ausdruck in einem Print-Job vorbereitet. Dem Inittialisierer der Klas- ▶



So präsentiert sich der *Printer Options*-Dialog (Bild 4)

PDF: Das vom Programm erzeugte PDF-Dokument (Bild 5)

se wird als Parameter die Variable mit dem Text übergeben. Im nächsten Schritt wird das Rendering des Textes durchgeführt. Hierzu bedienen wir uns der Klasse *UIPrintPageRenderer*.

Konfiguration des Ausdrucks

Nach dem Anlegen der entsprechenden Instanz werden die *UIMarkupTextPrintFormatter*-Instanz und die Start-Position, an der mit dem Rendering begonnen werden soll, übergeben. Dann geht es an die weitere Konfiguration des Ausdrucks. Dazu werden dessen Abmessungen festgelegt. Im Beispiel wird ein A4-Ausdruck mit 72 dpi zugrunde gelegt. Auch die zur Dimensionierung generierten Variablen *a4Page* und *rectangle* werden der Klasse *UIPrintPageRenderer* übergeben.

Das im Speicher generierte PDF-Dokument wird in einem Objekt vom Typ *NSMutableData data* abgelegt. Der Eigenschaft *numberOfPages* kann entnommen werden, wie viele Seiten das PDF-Dokument enthalten wird. Diese Eigenschaft wird nun in einer *for*-Schleife verwendet, um jede einzlne Seite zu formatieren. Nach der Generierung des PDF-Dokuments wird dieses im hier erläuterten Beispiel in einer Datei im temporären Verzeichnis der App gespeichert. Von dieser Stelle aus kann das Dokument dann geöffnet und beispielsweise in einer UIWebView angezeigt werden (Bild 5).

Fazit

Dass iOS-Geräte nicht nur zum Konsumieren von Inhalten, sondern auch zur Bearbeitung und Ausdruck von Dokumenten verwendet werden können, versteht sich von selbst. In diesem Artikel wurde beschrieben, wie entsprechende Funktionen in der eigenen App realisiert werden können. ■

Christian Bleske
ist Autor, Trainer und Entwickler mit dem Schwerpunkt Client/Server und mobile Technologien. Sein Arbeitsschwerpunkt liegt auf Microsoft-Technologien.
cb.2000@hotmail.de

Listing 2: PDF und HTML

```
import UIKit
class ViewController: UIViewController {
  @IBOutlet var uiWebView: UIWebView!
  override func viewDidLoad() {
    super.viewDidLoad()
    let htmlTitle =
    "<h1>Excepteur sint occaecat</h1></br>"
    let htmlText = "<b>Lorem ipsum dolor sit er elit
    lamet, consectetaur cillium adipisicing pecu, sed
    do eiusmod tempor magna aliqua.</b>"
    var completeText = htmlTitle
    completeText += htmlText
    let uiMarkupTextPrintFormatter =
    UIMarkupTextPrintFormatter(markupText:
    completeText)
    let uiPrintPageRenderer = UIPrintPageRenderer()
    uiPrintPageRenderer.addPrintFormatter(
    uiMarkupTextPrintFormatter,
    startingAtPageAtIndex: 0)
    let a4Page = CGRect(x: 0, y: 0, width: 595.2,
    height: 841.8) //
    let rectangle = CGRectInset(a4Page, 0, 0)
    uiPrintPageRenderer.setValue(NSValue(CGRect:
    a4Page), forKey: „paperRect“)
    uiPrintPageRenderer.setValue(NSValue(CGRect:
    rectangle), forKey: "printableRect")
    let data = NSMutableData()
    UIGraphicsBeginPDFContextToData(data,
    CGRectZero, nil)
    for i in 1...uiPrintPageRenderer.numberOfPages() {
      UIGraphicsBeginPDFPage();
      let bounds = UIGraphicsGetPDFContextBounds()
      uiPrintPageRenderer.drawPageAtIndex(i - 1,
      inRect: bounds)
    }
    UIGraphicsEndPDFContext();
    let path = "\(NSTemporaryDirectory())file.pdf"
    data.writeToFile(path, atomically: true)
    print("open \(path)")
    let url = NSURL.fileURLWithPath(path)
    let request = NSURLRequest(URL: url)
    uiWebView.loadRequest(request)
  }
}
```

DDC .NET Developer Conference 2016

- **05.12.2016 – DevSessions:**
  2 halbtägige Workshops
- **06.12.2016 – Konferenz:**
  1 Keynote, 2 Themenstränge,
  12 Vorträge
- **07.12.2016 – Workshops**
  Ganztägiges Praxistraining

Köln, Pullman Cologne

inklusive
SMART DATA
Developer Conference
Big Data & Smart Analytics

Trends, Lösungen und Know-how für für Profi-Entwickler

Themenauswahl:

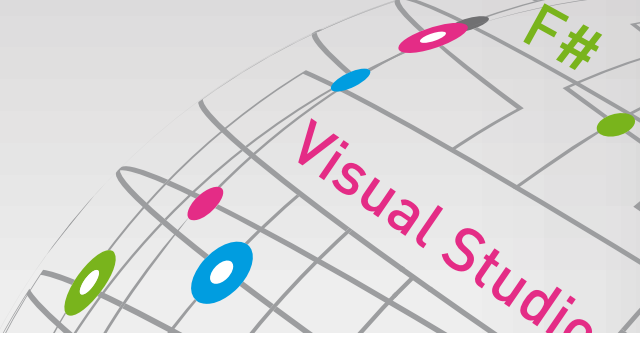
- **Softwarequalität:**
  Unit Testing, DDD, Refactor
- **Core:** .NET Core, Strings,
  Async/Await
- **Frontend:** UI Engineering,
  WPF, ASP.NET Core
- **Any App:** UWP, Xamarin,
  Microservices

dotnet-developer-conference.de **#netdc16** **DDConference**

Veranstalter: developer media Neue Mediengesellschaft Ulm mbH

Präsentiert von: dnp dotnetpro

.NET Developer Conference 2016



DevSessions: 05. Dezember 2016

	Frontend	Frontend	Softwarequalität	Architektur
09.00 – 13.00*	MVVM-Pattern mit WPF *Bernd Marquardt* Level 200	UWP Apps *André Krämer* Level 100	Software verbessern - mit aim42 *Gernot Starke* Level 100	Mikroservice Architektur mit Azure Service Bus *Torsten Helmich* Level 200
13.00 – 14.00	Mittagspause			
	Frontend	**Frontend**	**Softwarequalität**	**Backend**
14.00 – 18.00*	The WPF First Aid Kit *David Würfel* Level 300	Dreamteam: ASP.NET Core und Angular 2.0 *Johannes Hoppe* Level 100	TDD für Testmuffel *Hendrik Lösch* Level 200	Plattformunabhängiger Datenzugriff mit Entity Framework Core 1.0 *Dr. Holger Schwichtenberg* Level 100

*10.30 – 11.00 und 15.30 – 16.00 Kaffeepause

Konferenz-Programm: 06. Dezember 2016

SMART DATA Developer Conference
Big Data & Smart Analytics

08.45	Begrüßung durch den Veranstalter		Begrüßung
09.00 – 09.55	**Keynote:** Welcome HoloLens • *Damir Dobric und Andreas Erben*		Success Story: **Industrie 4.0 & Big Data**
	Softwarequalität	**Frontend**	**SMART DATA**
10.00 – 10.55	Fachliche Architektur mit DDD für .NET *Henning Schwentner* Level 200	Windows 10 UI Engineering für WPF-Entwickler *Thomas Immich* Level 300	Datenqualität: Big Data Matching *Dr. Hanna Köpcke*
11.00 – 11.30	Kaffeepause		Kaffeepause
11.30 – 12.25	12 tips for unit tests that don't cripple your codebase *Dennis Doomen* Level 200	Datenbindung Deluxe – Deep Dive in das Binding von WPF *Christian Giesswein* Level 400	4x4: Vier Real-World Beispiele bei vier Cloud Providern *Danny Linden*
12.30 – 13.25	Refactoring C# Legacy Code *Stefan Lieser* Level 300	Running your ASP.NET Core app in a Docker container *Maurice de Beijer* Level 200	NoSQL: Einführung in Graphdatenbanken mit Neo4j *Tobias Trelle*
13.30 – 14.30	Mittagspause		Mittagspause
14.00 – 14.30	Lunchsession		Recht: „German Cloud“ *Dobric und Erben*
	Core	**Any App**	**SMART DATA**
14.30 – 15.25	.NET Core *David Tielke* Level 200	UWP - New Horizons *Marek Pohanka und Daniel Bauer* Level 200	Recommender-Algorithmen mit R, Spark DataFrames und Spark MLlib *Henrik Behrens*
15.30 – 16.00	Kaffeepause		Kaffeepause
16.00 – 16.55	Async/Await: Out of Context *Tim Bussmann* Level 200	Xamarin - .NET für die Hosentasche *Sven-Michael Stübe* Level 200	Smart Analytics: Streaming mit Apache Flink *Stephan Papp*
17.00 – 18.00	Die dunkle Wahrheit über Strings im .NET Bereich *Christian Giesswein* Level 400	Microservices and Azure Service Fabric *Johannes C. Dumitru* Level 300	Visual, Active & Mobile Reports nach HICHERT®SUCCESS mit IBM Cognos *Holger Gerhards*
18.30 – 20.30	**Abendveranstaltung** mit Night Coding		

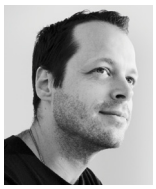
Programmänderung vorbehalten

Ihre Referenten (u.a.)

Daniel Bauer,
Kupferwerk GmbH

Damir Dobric,
DAENET Corporation

Dennis Doomen,
.NET Architect, Coach and Writer



Andreas Erben,
DAENET Corporation

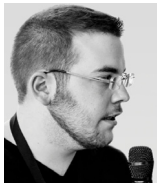
Christian Giesswein,
Giesswein Software-Solutions

Torsten Helmich,
SW Architekt und Scrum Master

Johannes Hoppe,
HAUS HOPPE ITS

Dr. Hanna Köpcke,
Webdata Solutions GmbH

Stefan Lieser,
Mitgründer der Clean Code Developer Initiative



Danny Linden,
OnPage.org

Stefan Papp,
The unbelievable Machine Company GmbH

Gernot Starke,
Gründungsmitglied des iSAQB e.V.

David Tielke,
david-tielke.de

Tobias Trelle,
codecentric AG

Ralf Westphal,
Mitgründer der Clean Code Developer Initiative

Workshops: 07. Dezember 2016

Workshop 1 | Moderne Anwendungsentwicklung mit .NET

David Tielke — Uhrzeit: 09.00 – 18.00 Uhr

In diesem Workshop verschafft Ihnen Trainer David Tielke einen Überblick über die verschiedensten Anwendungsarten: moderne Desktopanwendung mit WPF, Datenzugriff mit dem Entity Framework, moderne Webanwendungen mit ASP.NET MVC und WebAPI, Windows 10 Universal Apps, Anwendung mit Windows IoT auf einem Raspberry PI, Anwendung in die Cloud bringen.

Workshop 2 | Architektur Deluxe mit C# und .NET

Christian Giesswein — Uhrzeit: 09.00 – 18.00 Uhr

In diesem Workshop geht es darum Merkmale von "gutem" Code zu erläutern und auch die Techniken. Zum Beispiel lernen Sie Dependency Injection und Inversion-Of-Control praxisnah für jedes Projekt kennen, damit auch Ihre Software innerhalb kürzester Zeit eine höhere Qualität aufweist.

Workshop 3 | TDD 2.0 – Refactoring unnötig, offensichtlich, unvermeidbar

Ralf Westphal — Uhrzeit: 09.00 – 18.00 Uhr

Lassen Sie sich überraschen, wie "grüner Code" einerseits schon refaktorisiert entstehen kann - oder andererseits die Refaktorisierung unvermeidbar wird. Einführung in "TDD as if you meant it" und "Informed TDD".

Training: 08. Dezember 2016

developer media Training | Softwarequalität

David Tielke — Uhrzeit: 09.00 – 18.00 Uhr

Als erfahrener Trainer zeigt Ihnen David Tielke, wie Sie Softwarequalität spielend einfach in Ihren eigenen Projekten unterbringen, was die ersten Schritte sind und wie Schritt-für-Schritt ein mächtiger, nachhaltiger und transparenter Prozess für alle Beteiligten aufgebaut wird. Dabei liegt der Fokus auf allen Arten der Softwarequalität: richtig guten Code schreiben, kontrollieren, Architektur, Toolunterstützung, Testing ...

10% Rabatt bei Kombi-Anmeldung mit DDC

Kooperationspartner (Stand 10.06.2016):

BDOA digitalbusiness Cloud & IoT ecommerce magazin Geschäftserfolg im Internet W&V

Foto: Shutterstock / McLek

EIGENE FRAMEWORKS FÜR IOS ERSTELLEN

Flexibler Code

Frameworks erhöhen die Flexibilität des eigenen Codes und verhindern Code-Repeating.

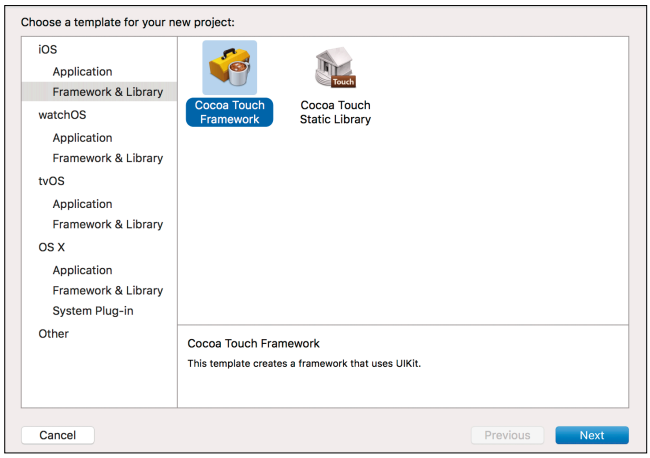
Als iOS-Entwickler hat man es ständig mit Frameworks zu tun. Sei es nun das Foundation-, das UIKit- oder das CoreData-Framework: Apple bietet eine Vielzahl an vorgefertigten Frameworks für die iOS-Entwicklung an, von denen manche – wie UIKit – unabdingbar für die tägliche Arbeit sind.

Diese Frameworks liefern grundlegende Klassen und Funktionen, die sich quer durch die verschiedensten iOS-Projekte hinweg verwenden und auf ganz unterschiedliche Art und Weise einsetzen lassen.

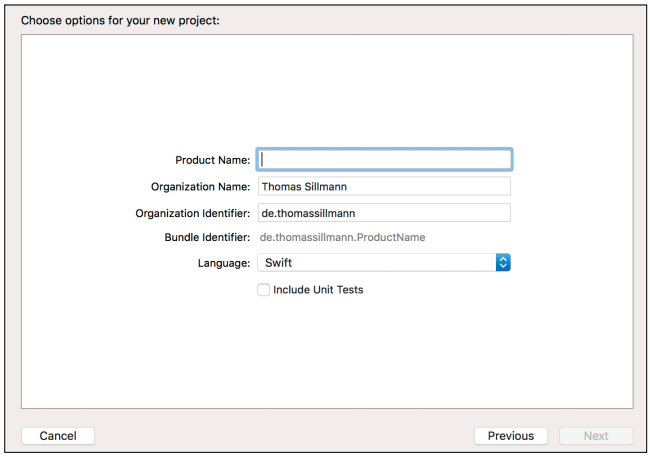
Darin besteht letztlich ja auch der große Vorteil und Nutzen von Frameworks: Anstatt bestimmten Code immer und immer wieder in allen möglichen Projekten und an verschiedensten Stellen neu zu schreiben oder hin und her zu kopieren, wird er stattdessen in ein Framework ausgelagert und kann darüber geteilt werden.

Wer bereits an mehreren verschiedenen iOS-Projekten gearbeitet hat, wird sich sehr wahrscheinlich an eigenen Code erinnern, der universal genutzt werden konnte und womöglich zur Wiederverwendung entweder immer wieder neu geschrieben oder per Copy and Paste in neue Projekte eingebunden wurde. Beide Verfahren sind dabei weder sonderlich schön noch komfortabel.

Das komplette Neuschreiben frisst enorm viel Zeit, und beim Kopieren sucht man zunächst einmal den ursprünglichen Code und muss ihn dann womöglich noch ein wenig anpassen und von Eigenheiten befreien, die explizit aufgrund des Ursprungsprojekts dort eingebunden wurden. Besser ist



Mit Hilfe der Vorlage Cocoa Touch Framework erstellt man ein neues iOS-Framework (Bild 1)



Frameworks werden zu Beginn ähnlich eingerichtet wie iOS-Projekte (Bild 2)

es, derartigen Code direkt von Beginn an in ein Framework zu packen und dann zukünftig für die gewünschte Funktionalität nur noch und ausschließlich eben jenes Framework zu pflegen.

Framework-Erstellung in Xcode

Doch wie erstellt man überhaupt ein Framework für iOS? Glücklicherweise ist das Erstellen von Frameworks bereits mit der Einführung iOS 8 und Xcode 6 deutlich komfortabler geworden. Seitdem findet sich beim Erstellen eines neuen Xcode-Projekts beziehungsweise eines neuen Targets innerhalb eines bestehenden Projekts eine neue Vorlage namens Cocoa Touch Framework (Bild 1).

Der Prozess zum Erstellen eines neuen Frameworks ist dabei sehr ähnlich zu dem beim Erstellen eines neuen iOS-Projekts. Man muss einen passenden Product und Organization Name sowie Organization Identifier definieren, aus dem sich dann automatisch ein eindeutiger Bundle Identifier zusammensetzt. Auch ein eigenes Unit-Test-Target kann direkt einem Framework hinzugefügt werden. Der entsprechende Assistent in Xcode führt durch diesen Schritt, bevor anschließend entweder das neue Framework-Projekt erstellt oder das neue Framework-Target einem bestehenden Projekt hinzugefügt wird (Bild 2).

Ein neu erstelltes Framework setzt sich dann aus insgesamt drei Dateien zusammen. Zunächst einmal ist da das eigentliche Framework-Target. Das kann als Äquivalent zu einem iOS-Target gesehen werden.

Es fasst alle Dateien und sonstigen Ressourcen des Frameworks zusammen, verfügt über konkrete Build Settings und besitzt eine eigene Deployment Info. Auch die Oberfläche zum Bearbeiten des Targets ist der zum Bearbeiten von iOS-Targets sehr ähnlich (Bild 3).

Daneben werden für das Framework von Xcode automatisch zwei erste Dateien erstellt: eine Objective-C-Header-Datei sowie eine *Info.plist*-Datei. Der Objective-C-Header wird übrigens selbst dann erstellt, wenn man als Programmiersprache beim Erstellen des Frameworks Swift ausgewählt hat, darüber braucht man sich in diesem Fall also nicht zu wundern.

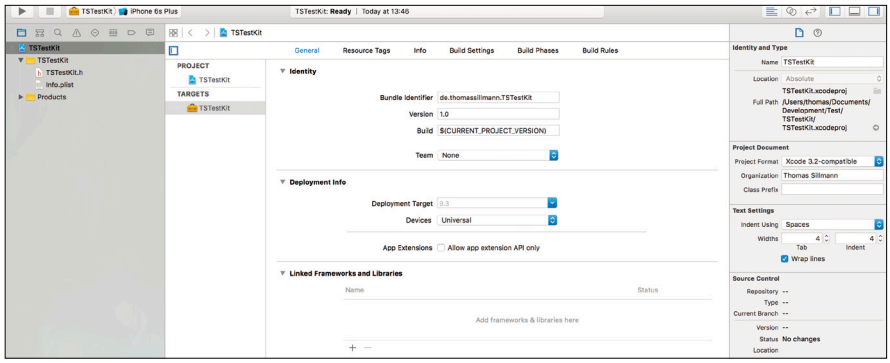
Die *Info.plist*-Datei auf der anderen Seite enthält grundlegende Informationen zum Framework und erfüllt dieselbe Aufgabe wie die *Info.plist* in iOS-Projekten, weshalb diese an dieser Stelle nicht weiter ausgeführt wird.

Der Framework-Header

Dem Objective-C-Header des Frameworks fällt eine ganz besondere Aufgabe zu: Dieser ist dafür verantwortlich, die Header all jener Klassen zu importieren, die durch das Framework von außen genutzt werden können.

Jede Klasse, die über das Framework ansprechbar sein soll, muss somit innerhalb des Framework-Header-Files mittels *#import*-Anweisung hinzugefügt werden. Fehlen sie an dieser Stelle, so stehen die entsprechenden Klassen nur innerhalb des Frameworks zur Verfügung und können deshalb ausschließlich für die internen Zwecke des Frameworks verwendet werden.

Auf diese Art und Weise lässt sich auch sehr gut zwischen öffentlichem und privatem Interface unterscheiden. Alle Eigenschaften und Methoden, die in den Headern der Klas- ▶



Die Einstellungen für ein Framework-Target ähneln stark denen von iOS-Targets (Bild 3)

Listing 1: Loggen von Nachrichten

```
// Header-Datei
#import <Foundation/Foundation.h>
@interface TSLogObjectiveC : NSObject
- (void)logMessage:(NSString *)message;
@end
// Implementation-Datei
#import „TSLogObjectiveC.h“
@implementation TSLogObjectiveC
- (void)logMessage:(NSString *)message {
    NSLog(@“Posted from Objective-C: %@“, message);
}
@end
```

sen deklariert sind, die im Framework importiert werden, können aufgerufen und verwendet werden. Alle anderen Eigenschaften und Methoden – beispielsweise diejenigen, die nur in der Implementierung einer Klasse auftauchen – können von außerhalb des Frameworks nicht angesprochen werden und sind sozusagen privat; nur innerhalb des Frameworks selbst können sie verwendet und angesprochen werden.

Mit diesen Headern gibt es aber ein Problem: Sie funktionierten ausschließlich mit Objective-C-Klassen. Das liegt daran, dass Objective-C explizit zwischen Header (der Deklaration der Eigenschaften und Methoden) und Implementierung (die eigentliche Logik zur Umsetzung der Methoden) unterscheidet. Swift-Klassen hingegen verfügen über keine separate Header-Datei. In Swift werden so Interface und Implementierung in einer Datei zusammengefasst. Was also bedeutet das für Entwickler?

Access Control in Swift

Erfreulicherweise ist die Lösung dieser Problematik relativ simpel. Bei Objective-C-Klassen muss entschieden werden, ob deren Header im Framework importiert werden soll und so nach außen hin zur Verfügung steht. Bei Swift-Klassen muss der Framework-Header hingegen überhaupt nicht angefasst werden.

Stattdessen wird innerhalb der Swift-Klasse selbst festgelegt, ob diese außerhalb des Frameworks zur Verfügung steht beziehungsweise welche Eigenschaften und Methoden einer Swift-Klasse nach außen zugänglich sind und welche nicht. In Swift wird dieses Verhalten über die sogenannte Access Control geregelt, die zwischen drei verschiedenen Levels unterscheidet: *public, internal* und *private*.

Jede Klasse und jede Methode sowie Eigenschaft in Swift, die ein Framework zur Verfügung stellen soll, damit sie auch von außerhalb des Frameworks genutzt werden kann, muss als *public* gekennzeichnet werden. Generell gehört damit *public* zu den wichtigsten Swift-Schlüsselwörtern bei der Entwicklung von iOS-Frameworks. Denn was nicht explizit als *public* deklariert wird, ist per Default *internal*. Das bedeutet, dass diese Klassen, Eigenschaften und Methoden nur innerhalb des eigentlichen Frameworks zur Verfügung stehen; von außerhalb des Frameworks (beispielsweise aus einem anderen Projekt heraus, welches das Framework importiert) können sie nicht aufgerufen und verwendet werden. *private* schließlich geht noch einen Schritt weiter und sorgt dafür, dass entsprechend gekennzeichnete Eigenschaften und Methoden nur innerhalb der Datei aufgerufen und verwendet werden können, in der sie deklariert wurden.

So weit einmal die grundlegende Theorie darüber, wie Frameworks Funktionen nach außen hin zur Verfügung stellen. Bei der Umsetzung gibt es noch das ein oder andere Detail zu beachten, weshalb im Folgenden ein konkretes kleines Beispielprojekt umgesetzt wird.

Logging mittels Objective-C und Swift

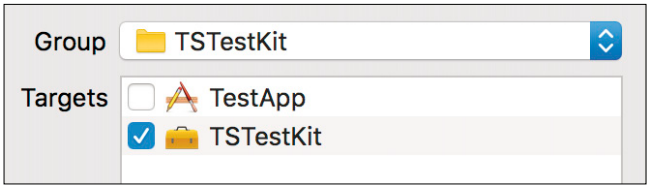
Im Folgenden wird ein Framework erstellt, das über zwei Klassen verfügt: eine Objective-C- und eine Swift-Klasse. Beide haben dieselbe Aufgabe: Sie sollen eine Methode zur Verfügung stellen, über die eine Nachricht auf der Konsole geloggt und ausgegeben wird.

Der Vorteil an diesem Beispiel – auch wenn oder gerade weil es von der Implementierung des Codes her sehr einfach gehalten ist – liegt darin, dass es sowohl den Umgang mit Objective-C- als auch mit Swift-Dateien zeigt und erklärt, worauf bei der Arbeit mit Frameworks jeweils zu achten ist.

Im ersten Schritt wird eine gewünschte Objective-C-Klasse erstellt. Sie trägt in diesem Beispiel den Namen *TSLogObjectiveC* und verfügt über eine einfache Methode *logMessage:*, die als Parameter einen String erwartet und diesen anschließend mittels NSLog über die Konsole ausgibt. Listing 1 zeigt die sehr überschaubare Implementierung dieser Klasse. Dabei ist in der Header-Datei die Methode *logMessage:* deklariert, in der Implementierung findet sich dann die zugehörige Umsetzung.

Diese Klasse wird dabei zunächst wie jede andere Datei dem Framework hinzugefügt, genau so, wie man es auch bereits von iOS-Projekten her kennt. Dabei ist darauf zu achten, bei der Wahl des Speicherorts auch das richtige Target auszuwählen, sofern das Framework nicht das einzige Target innerhalb des Xcode-Projekts ist (Bild 4).

Ein weiteres Detail, auf das man achten muss, ist der Status der sogenannten Target Membership. Normalerweise braucht man sich darüber keine Gedanken zu machen, da diese so eingestellt ist, dass sie für abgeschottete iOS-Projekte hervorragend funktioniert. Ähnlich wie die unter Swift verfügbaren Access Level regeln nämlich die drei verfügbaren



Achten Sie darauf, das richtige Target auszuwählen, wenn Ihr Framework ein Target unter mehreren in Ihrem Xcode-Projekt ist (Bild 4)

Optionen *Public, Private* und *Project* die Verfügbarkeit eines Objective-C-Headers innerhalb eines Projekts. Der Standard ist *Project* und sagt aus, dass die Deklaration eben jenes Headers innerhalb des abgeschotteten Projekts zur Verfügung steht, aber nicht darüber hinaus. Genau Letzteres soll nun ja aber mit einem Framework umgangen werden, weshalb jene Option auf *Public* gesetzt werden muss.

Den aktuellen Status der Target Membership kann man einsehen, indem man die Header-Datei der Objective-C-Klasse auswählt und in der Utilities Area in den File Inspector wechselt. Der gleichnamige Fensterabschnitt *Target Membership* zeigt zum einen, zu welchem Target diese Klasse gehört (hier sollte das Framework angehakt sein), und zum anderen, mit welchem Status sie in diesem Framework hinterlegt ist. Steht hier etwas anderes als *Public* und soll der Header auch außerhalb des Frameworks verwendet werden können, so muss an dieser Stelle zwingend *Public* ausgewählt und gesetzt werden (Bild 5).

Damit nun abschließend die Objective-C-Klasse auch tatsächlich außerhalb unseres Frameworks genutzt und verwendet werden kann, muss noch die Header-Datei der Klasse im Objective-C-Framework-Header importiert werden. Dazu öffnet man jene Framework-Header-Datei und fügt am Ende eine neue *#import*-Anweisung hinzu. Diese ist dabei wie folgt aufgebaut:

```
#import <Framework-Name/Klassenname.h>
```

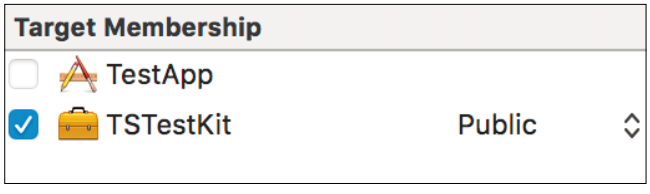
In diesem Beispiel lautet der Name des Frameworks (dabei handelt es sich um den *Product Name,* der bei der Erstellung des Frameworks festgelegt wurde) *TSTestKit*, während die Objective-C-Klasse den Namen *TSLogObjectiveC* trägt. Entsprechend lautet die *#import*-Anweisung, die dem Framework-Header hinzugefügt werden muss, um die Klasse *TSLogObjectiveC* verfügbar zu machen, wie folgt:

```
#import <TSTestKit/TSLogObjectiveC.h>
```

Damit steht die Klasse *TSLogObjectiveC* mit allen in der Header-Datei deklarierten Eigenschaften und Methoden in jedem Projekt bereit, welches das Framework importiert.

Erstellen und Verwenden einer Swift-Klasse

Bei der Arbeit mit Swift in einem Cocoa Touch Framework gibt es ein paar Unterschiede zu beachten. Der erste große Unterschied wurde bereits genannt: Da Swift-Klassen keine



Soll die Deklaration eines Objective-C-Headers außerhalb des Frameworks zur Verfügung stehen, so muss in der Target Membership die Option *Public* ausgewählt werden (Bild 5)

Listing 2: Swift-Klasse zum Loggen von Nachrichten

```
import Foundation
public class TSLogSwift {
  public var logMessage: String
  public init(message: String) {
    logMessage = message
  }
  public func printLogMessage() {
    print(„Posted from Swift: \(logMessage)“)
  }
}
```

Unterscheidung zwischen einer Header- und einer Implementation-Datei kennen, muss die Access Control regeln, welche Klassen, Eigenschaften und Methoden öffentlich auch außerhalb des Frameworks ansprechbar und verwendbar sind und welche nicht.

Das hat ebenso zur Folge, dass Swift-Klassen – selbst wenn sie öffentlich auch außerhalb des Frameworks zur Verfügung stehen sollen – nicht im Framework-Header importiert werden; schließlich gibt es keine zugehörige separate Header-Datei bei Swift-Klassen, dieser Schritt entfällt also immer bei Swift.

Klasse erhält einen Initializer

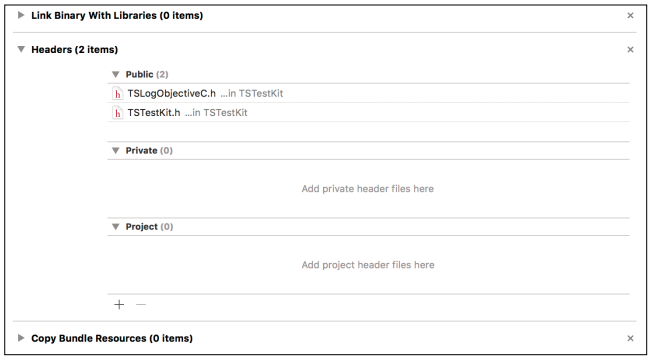
Bevor man sich aber mit diesen Details auseinandersetzt, geht es zunächst einmal an das Erstellen einer Swift-Datei für das Framework. Auch das wird auf dieselbe Art und Weise umgesetzt wie in iOS-Projekten. In diesem Beispiel wird die Implementierung der neu zu erstellenden Swift-Klasse aber ein wenig anders gelöst als unter Objective-C.

In Swift erhält die Klasse einen Initializer, der einen String als zu loggende Nachricht entgegennimmt und in einer Property speichert. Es steht dann eine einfache Methode namens *printLogMessage* bereit, die eben jene bei der Initialisierung eines Objekts der Klasse übergebene Nachricht in der Konsole ausgibt. Listing 2 zeigt die Implementierung dieser zusätzlichen Klasse namens *TSLogSwift* innerhalb des *TSTestKit*-Frameworks.

Hierbei fällt bereits der exzessive Gebrauch des Schlüsselworts *public* auf. In diesem Beispiel sind alle Eigenschaften und Methoden sowie die Klasse selbst als *public* deklariert. Das bedeutet, dass sie auch außerhalb des Frameworks genutzt werden können, also beispielsweise in einem Dritt-Projekt, welches das Framework importiert.

Würde man stattdessen das Schlüsselwort *private* verwenden, könnten alle Eigenschaften und Methoden nur ausschließlich innerhalb der Swift-Datei verwendet werden, in der eben jener Code steht. Das Weglassen jeglicher Access-Level-Schlüsselwörter setzt den Access Level automatisch auf *internal*, wie es auch in der Regel bei iOS-Projekten für die meisten Eigenschaften und Methoden verwendet wird. Es würde dafür sorgen, dass zwar die Klasse selbst mit all ihren Funktionen innerhalb des gesamten Frameworks, nicht ▶

Vielleicht fehlt die Möglichkeit, eine Option für die Target Membership festzulegen (Bild 6)



In den Build Phases lassen sich die zu einem Target zugeordneten Header einsehen (Bild 7)

aber außerhalb des Frameworks (beispielsweise für Dritt-Projekte) verwendet werden kann.

Eine so erstellte Swift-Klasse mit Verwendung des Schlüsselworts *public* (zumindest für all jene Eigenschaften und Methoden, die tatsächlich auch außerhalb des Frameworks zur Verfügung stehen sollen) ist prinzipiell bereits vollständig vorbereitet, um im Framework verwendet werden zu können. Im Gegensatz zu Objective-C-Klassen ist ein Importieren des (sowieso nicht existierenden) Headers innerhalb des Framework-Headers nicht notwendig.

Inkonsistentes Verhalten

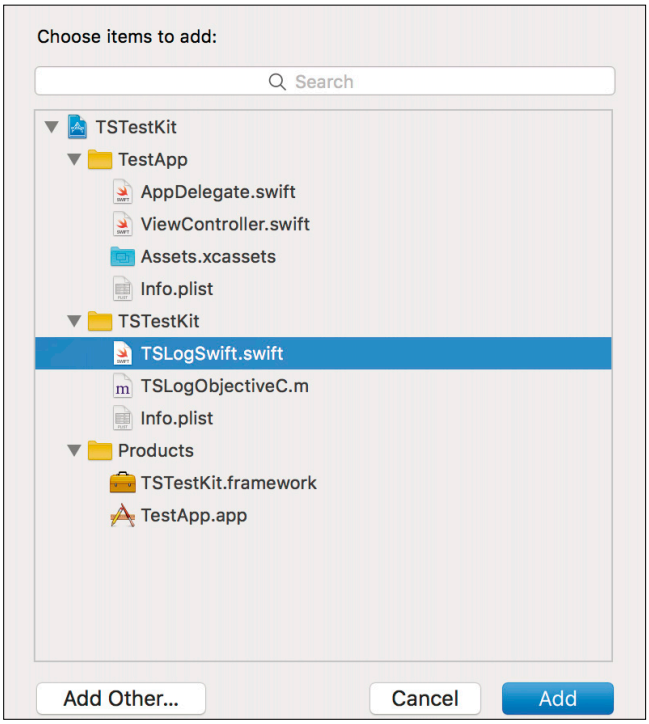
Allerdings muss auch für eine Swift-Klasse in der Target Membership die Option *Public* gesetzt werden, damit die Klasse auch tatsächlich außerhalb des Frameworks aufgerufen und verwendet werden kann. Und hierbei habe ich unter Xcode 7.3.1 bisweilen ein sehr inkonsistentes Verhalten festgestellt, auf das ich auch an dieser Stelle hinweisen möchte.

Wirft man einen Blick in die Target Membership einer Swift-Datei, ist zwar zu erkennen, zu welchem Target (in diesem Fall dem Framework) die Datei gehört, es fehlt aber die Auswahlmöglichkeit für die Option, die festlegt, ob die Datei nun Public, Private oder Project ist (Bild 6). Sollte das der Fall sein, so muss die Datei noch händisch mit der passenden Option dem Framework-Target hinzugefügt werden.

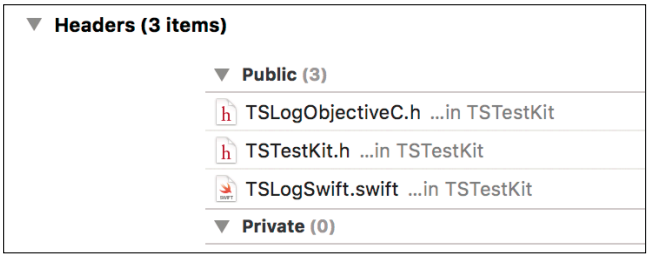
Um das durchzuführen, wählt man in der Projektansicht das Framework-Target aus und wechselt in den Reiter *Build Phases*. Dort findet sich ein Abschnitt namens *Headers* (hinter dem Namen steht in Klammern noch die Anzahl der zugeordneten Header zu diesem Target). Klappt man diesen auf, werden drei verschiedene Tabellen mit den Titeln *Public, Private* und *Project* aufgeführt und angezeigt; genau jene Optionen, die wir in der Target Membership setzen können (Bild 7). Fehlt dort die gewünschte Datei (in diesem Beispiel die *TSLogSwift.swift*-Datei), so muss diese mittels der Plus-Schaltfläche am unteren Rand als Header hinzugefügt werden. Über diese Schaltfläche öffnet sich eine Projektübersicht, aus der die gewünschte Klasse ausgewählt und per Klick auf die Schaltfläche *Add* dem Target als Header hinzugefügt werden kann (Bild 8).

Standardmäßig landen so einem Target hinzugefügte Header in der Tabelle *Project*. Soll der Header stattdessen als *Private* oder *Public* verwendet werden, so muss er mittels Drag and Drop in die entsprechende Tabelle verschoben werden. In diesem Beispiel wird somit die Datei *TSLogSwift.swift* von der *Project*- in die *Public*-Tabelle gezogen (Bild 9).

Zwar ist nach dem händischen Hinzufügen eines Headers zu einem Target auf die beschriebene Art und Weise bei Auswahl der zugehörigen Datei in der Target Membership noch immer nicht ersichtlich, welche Option dafür ausgewählt wurde, dennoch ist der Header damit korrekt im Target eingebunden.



Neue Header können aus der Projektübersicht heraus einem Target hinzugefügt werden (Bild 8)



TSLogSwift.swift ist nun als Public Header im Target hinterlegt (Bild 9)

Um ein Framework in einem iOS-Projekt verwenden zu können, muss es dort als Embedded Binary hinzugefügt werden (Bild 10)

Ist ein Framework bereit, geteilt und in anderen Projekten verwendet zu werden (so wie das in diesem Artikelbeispiel erstellte einfache *TSTestKit*-Framework), ist der einfachste Weg, es an anderer Stelle einzubinden, der, es zu kopieren.

Idealerweise legt man das Framework als eigenständiges Projekt an einer zentralen Stelle ab und referenziert von iOS-Projekten darauf, die dieses Framework verwenden möchten. Das hat auch den positiven Nebeneffekt, dass Neuerungen und Verbesserungen am Framework direkt in allen Projekten, die dieses einbinden, zur Verfügung stehen.

Einbinden und Aufrufen eines Frameworks

In diesem Beispiel gehe ich den umgekehrten Weg und füge dem Framework-Projekt ein iOS-Target hinzu, um es darüber einmal zu testen und die Einbindung des Frameworks in einem richtigen Projekt zu prüfen. Als Vorlage für das Target verwende ich dabei eine Single View Application.

Innerhalb des damit automatisch von Xcode erzeugten zentralen ViewControllers soll nun die Funktionalität des Frameworks getestet werden. Aus diesem Grund werden zwei Methoden erstellt; die eine arbeitet mit der *TSLogObjectiveC*-, die andere mit der *TSLogSwift*-Klasse. Beide Methoden werden dann aus der *viewDidLoad*-Methode des ViewControllers heraus aufgerufen.

Listing 3: Framework in einem iOS-Projekt

```
import UIKit
import TSTestKit
class ViewController: UIViewController {
    override func viewDidLoad() {
        super.viewDidLoad()
        postFromObjectiveC()
        postFromSwift()
    }
    private func postFromObjectiveC() {
        let objectiveCLog = TSLogObjectiveC()
        objectiveCLog.logMessage(„A log message.“)
    }
    private func postFromSwift() {
        let swiftLog = TSLogSwift
        (message: „Another log message.“)
        swiftLog.printLogMessage()
    }
}
```

Um überhaupt das Framework innerhalb einer Objective-C- oder Swift-Datei verwenden zu können, muss es zunächst einmal in das Projekt eingebunden und anschließend in der Klasse, in der es verwendet werden soll, importiert werden. Dazu wechselt man zunächst in die Target-Einstellungen jenes Targets, das das Framework nutzen soll, und wechselt dort auf den Reiter *General*. Dort findet sich ein Abschnitt *Embedded Binaries*. Hier muss nun per Klick auf die *Plus*-Schaltfläche das Framework-Target aus der Projektliste ausgewählt und per Klick auf die Schaltfläche *Add* dem Ziel-Target hinzugefügt werden (Bild 10).

Im nächsten Schritt geht es um den Import des Frameworks in den Quellcode-Dateien, in denen entsprechende Funktionen genutzt werden sollen. Unter Objective-C wird dabei die folgende Import-Anweisung verwendet, idealerweise innerhalb der Implementierung der gewünschten Klasse:

```
#import <TSTestKit/TSTestKit.h>
```

Darüber wird der zentrale Framework-Header importiert, und damit auch all jene Header, die darin vom Framework hinterlegt sind (wie eben der Header für die *TSLogObjectiveC*-Klasse). In Swift sieht diese Importanweisung ein wenig schlanker aus:

```
import TSTestKit
```

Listing 3 zeigt nun die beispielhafte Implementierung zweier Methoden innerhalb der ViewController-Klasse des iOS-Targets, die jeweils eine Instanz der Klasse *TSLogObjectiveC* beziehungsweise *TSLogSwift* erstellen und die dazugehörige Funktionalität verwenden, um eine Nachricht über die Konsole auszugeben (dieses Listing ist rein in Swift geschrieben).

Benennung von Frameworks und deren Klassen

Erstellt man ein eigenes Framework, sollte man sich frühzeitig einige grundlegende Gedanken zur Benennung machen. Im Gegensatz zu typischen iOS-Projekten, die eher für sich alleine stehen und nicht mit anderen Projekten geteilt werden, ist es gerade die Aufgabe eines Frameworks, in vielen verschiedenen Projekten verwendet zu werden. Besitzen nun Klassen eines Frameworks sehr generische Namen wie ViewController oder DatabaseManager, so kann es möglicherweise schnell zu Konflikten mit einem Projekt kommen, in dem das Framework eingebunden wird, weil es dort Klassen mit exakt demselben Namen gibt.

Um dieses Problem bestmöglich zu vermeiden, sollten Klassen eines Frameworks immer über ein Präfix verfügen. Dieses Präfix setzt sich typischerweise aus zwei Großbuchstaben zusammen.

In dem gezeigten Beispiel habe ich das bereits so entsprechend umgesetzt und für die beiden Klassen das Präfix *TS* verwendet. Welches Präfix man dabei letztendlich verwendet, ist jedem selbst überlassen, wobei man nach Möglichkeit darauf achten sollte, nicht bereits von Apple genutzte Präfixe wie *UI* oder *NS* zu verwenden. Das verwendete Präfix sollte sich dann ebenso bereits im Namen des Frameworks ▶

niederschlagen, um ein einheitliches Gesamtbild zu schaffen (wie im gezeigten Beispiel mit dem Framework-Namen *TSTestKit)*.

Vorteile und Einsatzzwecke von Frameworks

Der große Vorteil von Frameworks liegt – wie bereits beschrieben – darin, dass Code mit deren Hilfe schnell und einfach zwischen mehreren grundverschiedenen Projekten geteilt werden kann. Sie mindern mögliche Code-Redundanz und fassen logische Einheiten von zusammengehörigen Funktionen in einem eigenen separaten Projekt zusammen.

Dabei müssen Frameworks nicht immer als komplett eigenständige Projekte entstehen. In der Praxis ist es durchaus sinnvoll, eigene Frameworks zunächst einmal innerhalb eines bestehenden iOS-Projekts zu erstellen und hinzuzufügen, ehe man sie komplett als eigenständige Projekte extrahiert. Stellt man so beispielsweise fest, das bestimmter Code immer und immer wieder in verschiedenen Projekten verwendet wird, so kann man diesen initial in einem dieser Projekte in einem zusätzlichen Framework-Target auslagern und darüber gleichzeitig testen, ob das Framework wie gewünscht arbeitet und alles noch wie vorgesehen funktioniert.

Ist das der Fall, kann dieses Framework dann als Basis für dieselbe Funktionalität in anderen Projekten verwendet werden. Dieser Ansatz führt auch direkt zu ersten Erfolgserlebnissen, wenn man nicht nur einfach ein Framework entwickelt, sondern dieses auch umgehend praktisch in einem Projekt zum Einsatz kommt. So können nach und nach verschiedene Frameworks entwickelt werden, um die Entwicklung von iOS-Projekten langfristig zu optimieren und zu beschleunigen.

Unterschied zur Cocoa Touch Static Library

Zum Abschluss dieses Artikels möchte ich noch auf den Unterschied zwischen einem beschriebenen Cocoa Touch Framework und einer Cocoa Touch Static Library eingehen (Letztere steht ebenfalls als erstellbares Target in Xcode zur Verfügung).

Generell besteht der größte technische Unterschied zwischen diesen beiden Elementen darin, wie sie mit iOS-Projekten zusammenspielen. Ein Cocoa Touch Framework ist ein eigenes Target, aus dem ein eigenständiges Build-Produkt hervorgeht (so wie bei iOS-Apps auch).

Dieses Produkt ist unabhängig von den iOS-Projekten, in denen es verwendet wird. Eine Static Library hingegen kann zwar auch – wie ein Cocoa Touch Framework – zwischen verschiedenen Projekten geteilt werden, es wird aber immer zusammen mit dem iOS-Projekt gebaut und direkt in dieses integriert; es ist also weniger eigenständig als ein Cocoa Touch Framework.

Generell sind Cocoa Touch Frameworks auch deutlich funktionaler ausgelegt und leistungsfähiger. So unterstützen diese beispielsweise auch App Extensions in iOS (im Gegensatz zu Cocoa Touch Static Libraries). Zudem sind Cocoa Touch Static Libraries rein auf Objective-C ausgelegt und unterstützen offiziell kein Swift. Das zeigt sich bereits bei der Erstellung einer solchen Static Library, bei der die Frage nach der gewünschten Programmiersprache entfällt und stattdessen direkt eine Objective-C-Klasse als Basis generiert wird.

Um eine höchstmögliche Flexibilität bei der Erstellung von und Arbeit mit Frameworks sicherzustellen, sollte man in der Regel lieber auf ein Cocoa Touch Framework als auf eine Cocoa Touch Static Library setzen.

Links zum Thema

- Apple-Dokumentation zum Thema Frameworks
  https://developer.apple.com/library/mac/documentation/MacOSX/Conceptual/BPFrameworks/Concepts/WhatAreFrameworks.html
- Video »Building Modern Frameworks« von der WWDC 2014
  https://developer.apple.com/videos/wwdc/2014/?id=416
- Programming with Objective-C
  https://developer.apple.com/library/mac/documentation/Cocoa/Conceptual/ProgrammingWithObjectiveC/Introduction/Introduction.html
- Programmiersprache Swift
  https://developer.apple.com/swift

Fazit

Mit dem Einzug der Cocoa Touch Frameworks in Xcode können iOS-Entwickler erstmals selbst umfangreiche und komplexe Frameworks für iOS erstellen. Die grundlegende Arbeit mit Frameworks ist dabei in vielen Aspekten identisch mit der mit iOS-Projekten, auch wenn es diverse Feinheiten zu beachten gilt (vorrangig den zentralen Framework-Header sowie die Target Membership der Framework-Dateien).

Unter Swift ist darüber hinaus das Schlüsselwort *public* bei der Deklaration all jener Klassen, Eigenschaften und Methoden zu beachten, die das Framework zur Verfügung stellt und die somit auch außerhalb des Frameworks genutzt werden können sollen. Idealerweise entstehen neue Frameworks im Zuge eines bestehenden iOS-Projekts. Das erlaubt es, direkt den Code eines solchen bestehenden Projekts zu optimieren und gleichzeitig das neu erstellte Framework auf korrekte Funktionalität hin zu testen. Es kann dann als Basis für weitere Projekte verwendet und dort eingebunden werden.

Für weitere Informationen lohnt sich ein Blick in Apples Dokumentation zum Thema Frameworks sowie das Video »Building Modern Frameworks« von der WWDC 2014. ■

Thomas Sillmann
ist iOS-App-Entwickler, Trainer und Autor. Freiberuflich tätig programmiert er für den App Store eigene Apps sowie Apps in Form von Kundenaufträgen. Er ist Autor eines erfolgreichen Fachbuchs und mehrerer Artikel in Fachzeitschriften.
www.thomassillmann.de

developer media

developer media Trainings

Updates für Ihr Know-how

„Erfolgreiche Entwickler haben den für ihre Software passenden Testmix. Lernen Sie, wie man externe wie interne Qualität effizient sicherstellt."

Sebastian Bergmann
PHP-Consultant und Trainer

Angular 2 mit TypeScript

Trainer: Johannes Hoppe, Gregor Woiwode

3 Tage, 29.-31.08.2016, München
Ab 2.199 EUR zzgl. MwSt.

Cross-Plattform-Apps mit C# und Xamarin

Trainer: Sebastian Seidel

3 Tage, 12.-14.07.2016, Köln
Ab 2.199 EUR zzgl. MwSt.

PHPUnit erfolgreich einsetzen

Trainer: Sebastian Bergmann

2 Tage, Köln, Termin nach Absprache
Ab 1.999 EUR zzgl. MwSt.

Codequalität mit JavaScript

Trainer: Golo Roden

3 Tage, Termin & Ort nach Absprache
Ab 2.399 EUR zzgl. MwSt.

Ihr Ansprechpartner:
Fernando Schneider – Key Account Manager – developer media
Telefon: +49 (0)89 74117-831 – E-Mail: fernando.schneider@developer-media.de

developer-media.de/trainings

GENERICS IN SWIFT

Dynamischer Code

Wie sich mit Hilfe von Generics dynamischer Code erstellen lässt.

Apples Programmiersprache Swift hat einige Besonderheiten und gelungene Sprachmerkmale zu bieten. Eines der mächtigsten Sprachfeatures überhaupt stellen dabei zweifelsohne die sogenannten Generics dar (Bild 1). Mit Generics ist es möglich, Funktionen und Typen zu definieren, die über einen oder mehrere dynamisch definierte Typen verfügen; so weit die grobe und durchaus kompliziert klingende Zusammenfassung von Generics in einem Satz. Tatsächlich sind Generics eben nicht zuletzt aufgrund ihrer besonderen Funktionsweise und der durchaus ungewohnten und etwas komplizierten Syntax alles andere als trivial. Daher erfahren Sie im Folgenden Schritt für Schritt, wie die Arbeit mit Generics funktioniert, welche Vorteile sich daraus für Sie ergeben und wann Generics an ihre Grenze stoßen.

Einsatzzweck von Generics

Um die Funktionsweise und den Nutzen von Generics am besten vorzustellen, lohnt der Blick in ein praktisches Beispiel. Als Vorlage soll dabei die Methode *swap(_:_:)* aus der Swift Standard Library dienen. Diese hat die Aufgabe, die Werte von zwei Objekten miteinander zu vertauschen. Das folgende Listing zeigt eine derartige, eigens von mir geschriebene Methode, die die Werte von zwei String-Objekten miteinander vertauscht:

```
func changeStringValue(inout firstValue: String, inout
withStringValue secondValue: String) {
  let temporaryFirstValue = firstValue
  firstValue = secondValue
  secondValue = temporaryFirstValue
}
```

Die Funktionsweise der gezeigten Methode *changeStringValue(_:withStringValue:)* ist dabei sehr simpel: Ihr werden zwei Parameter vom Typ *String* übergeben, die mittels *inout* gekennzeichnet sind. Letzteres hat zur Folge, dass die Werte direkt in den aufrufenden Objekten der Methode entsprechend vertauscht werden (und nicht nur innerhalb der Methode selbst). Ohne *inout* wäre eine Änderung der Parameter der Methode gar nicht möglich.

Die Methode speichert den Wert des ersten zu tauschenden Parameters in einer neu erstellten temporären Konstante *temporaryFirstValue* zwischen und weist dem ersten Parameter anschließend den Wert des zweiten Parameters zu. Anschließend wird dem zweiten Parameter der ursprüngliche Wert des ersten Parameters zugewiesen, der in eben jener temporären Konstante *temporaryFirstValue* enthalten ist. Das ist alles gut und richtig so.

Allerdings besitzt diese Methode eine starke Einschränkung: Sie funktioniert nur mit Parametern vom Typ *String*. Möchte man nun auf genau dieselbe Art und Weise *Integer, Double* oder sonstige Typen miteinander vertauschen, müsste man nach dem jetzigen Stand für jeden einzelnen Typ eine eigene Methode erstellen.

Das folgende Listing zeigt dieses Prozedere für zwei entsprechende Methoden, über die einmal Integer und einmal Double miteinander vertauscht werden können:

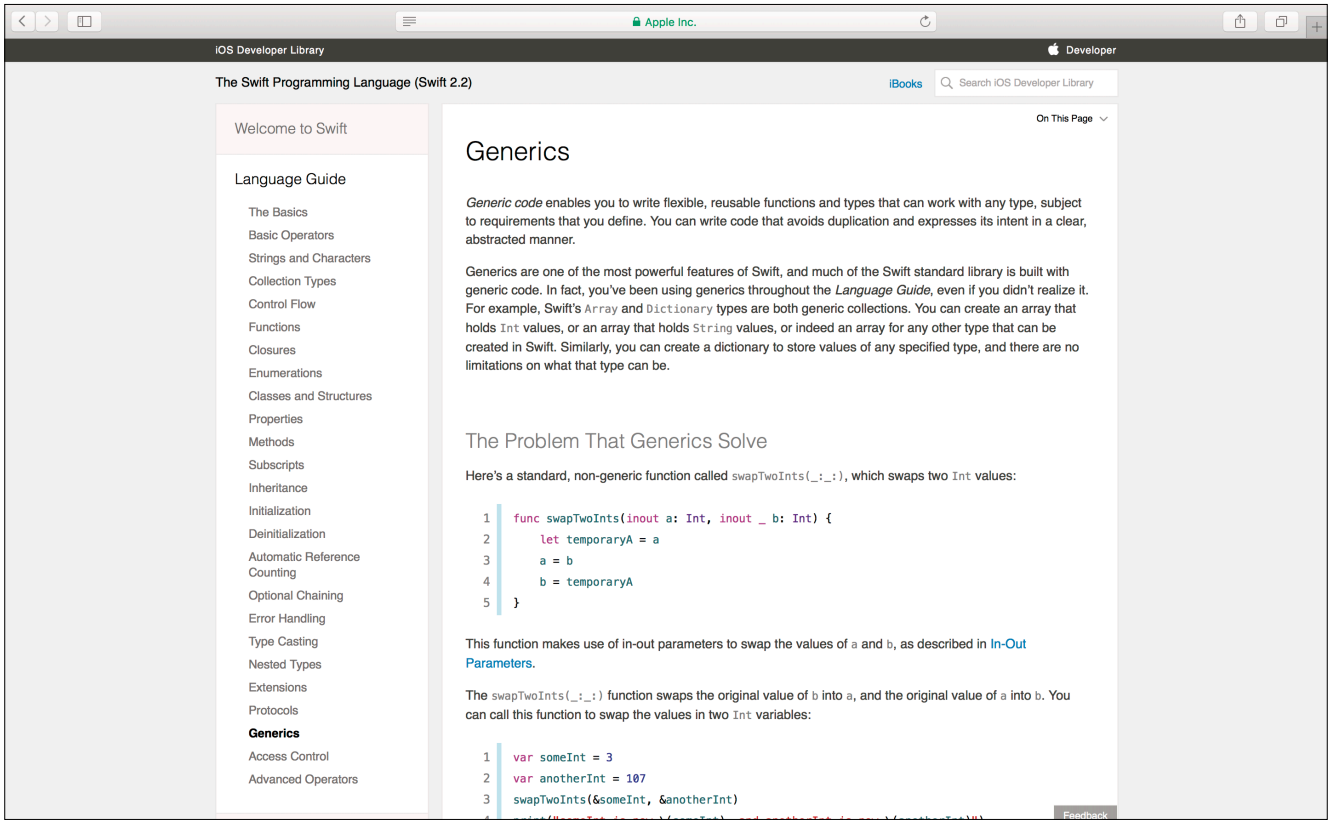
```
func changeIntValue(inout firstValue: Int, inout
withIntValue secondValue: Int) {
  let temporaryFirstValue = firstValue
  firstValue = secondValue
  secondValue = temporaryFirstValue
}
func changeDoubleValue(inout firstValue: Double, inout
withDoubleValue secondValue: Double) {
  let temporaryFirstValue = firstValue
  firstValue = secondValue
  secondValue = temporaryFirstValue
}
```

Betrachtet man die insgesamt drei Methoden aus den beiden Beispielen, so fällt auf, dass alle Methoden über exakt dieselbe Implementierung verfügen. Der einzige Unterschied besteht im Typ der Parameter, die sie jeweils besitzen; einmal ist es *String,* einmal *Int* und einmal *Double*. Doch unabhängig davon wird – ganz gleich, um welchen Typ es sich handelt – immer derselbe Code ausgeführt.

Genau diese Problematik – Code, der unabhängig von einem ganz bestimmten Typ funktioniert – ist prädestiniert für Generics. Eingangs habe ich zur Kurzbeschreibung von Generics geschrieben: »Mit Generics ist es möglich, Funktionen und Typen zu definieren, die über einen oder mehrere dynamisch definierte Typen verfügen.« Genau das machen wir uns jetzt zunutze. Ideal wäre eine Methode, die statt eines statischen Typs für die Parameter unserer Methode wie String, Int oder Double einen generischen Typ erwartet. Was dieser generische Typ genau ist – *String, Int, Double* oder etwas ganz anderes – wird dann dynamisch beim Aufruf der Methode entschieden.

Deklaration einer Generic Function

Eine Funktion, die einen oder mehrere dynamische Typen für Parameter erwartet, legt für eben jene Parameter Platzhalter bei der Deklaration der Funktion fest. Diese Platzhalter werden zwischen einem Kleiner- und Größerzeichen kommase-



Generics gehören zu den mächtigsten und vielseitigsten Sprachmerkmalen von Swift (Bild 1)

pariert voneinander angegeben, bevor die eigentlichen Funktionsparameter in runden Klammern folgen.

Wie diese Platzhalter benannt werden, ist dabei dem Entwickler überlassen. In der Regel sollten bei mehreren Platzhaltern verständliche und beschreibende Namen gewählt werden, die genauer erklären, wofür ein Platzhalter-Typ steht. Bei einzelnen Platzhaltern haben sich hingegen simple Einbuchstaben-Bezeichungen wie *T, U* oder *V* etabliert.

Da die Beschreibung des Aufbaus einer Generic Function etwas kompliziert klingt, zeigt der folgende Code zunächst einmal lediglich die Deklaration einer solchen, die über einen einzigen dynamischen Typ verfügt, der als *T* definiert wird. Als Basis für die Deklaration nehme ich das vorherige Beispiel zum Austausch von zwei Werten:

```
func changeValue<T>(...) {
  ...
}
```

Zunächst einmal besteht der einzige Unterschied der Generic Function im Vergleich zu den vorherigen Beispielen im <*T*>, das nach dem Funktionsnamen und vor den Parametern innerhalb der runden Klammern gesetzt ist.
Wie beschrieben werden innerhalb dieses Kleiner- und Größerzeichens die gewünschten dynamischen Typen für diese Funktion definiert, in diesem Beispiel also ein einziger dynamischer Typ *T*.

Ein so definierter dynamischer Typ steht innerhalb der Funktion, in der er deklariert wurde, zur Verfügung, ganz gleich ob für Parameter oder innerhalb der Implementierung der Funktion. Um welchen konkreten Typ es sich bei *T* letztendlich handelt, ist bei einer Generic Function von deren Aufruf abhängig. Dazu muss eine Generic Function für jeden definierten dynamischen Typ mindestens einen Parameter erwarten, der genau diesem Typ entspricht. Wird nun die Generic Function aufgerufen, wird *T* in genau jenen Typ umgewandelt, der dem entsprechenden Typ des übergebenen Parameters der Methode entspricht.

Das folgende Beispiel zeigt dazu einmal die vollständige Deklaration und Implementierung einer Generic Function:

```
func changeValue<T>(inout firstValue: T, inout withValue
secondValue: T) {
  let temporaryFirstValue = firstValue
  firstValue = secondValue
  secondValue = temporaryFirstValue
}
```

Es handelt sich dabei um die Weiterentwicklung der Generic Function aus dem vorigen Beispiel. Die Methode nimmt nun – wie aus den eingangs gezeigten Beispielen bekannt – zwei Parameter entgegen, denen beiden der dynamische Typ *T* zugewiesen wird. Der Rest der Implementierung entspricht den bereits bekannten Beispielen.

Rein äußerlich hat sich bei dieser Funktion nicht viel verändert. Tatsächlich besteht auch der einzige Unterschied im Vergleich zu den vorherigen Beispielen darin, dass sie einen dynamischen Typ *T* definiert und diesen als Typ für die bei- ▶

den Parameter *firstValue* und *secondValue* verwendet. Diese kleine Änderung hat aber enorme Auswirkungen auf die Funktionalität und Verwendungsmöglichkeit dieser Funktion. Denn nun können ihr zwei Parameter jedes beliebigen Typs übergeben werden, solange die beiden Parameter nur demselben Typ entsprechen. Ganz gleich ob die Funktion nun mit zwei Strings, zwei Integern oder zwei Double aufgerufen wird, sie wird immer funktionieren und die Werte der beiden Parameter vertauschen.

In gewisser Weise kann man sich die Funktionalität dieser Generic Function wie folgt vorstellen: Beim Aufruf der Funktion sieht Swift, dass der erste Parameter vom dynamischen Typ *T* ist. Was für ein konkreter Typ das ist, definiert Swift anhand des Typs des Parameters, der der Methode für *firstValue* übergeben wird. Handelt es sich dabei um einen Integer, wird für *T* der Typ *Integer* verwendet, handelt es sich um einen String, wird für *T* stattdessen String eingesetzt (Bild 2).

Das bedeutet aber auch, dass der zweite Parameter der Methode, der ebenfalls vom dynamischen Typ *T* ist, folgerichtig auch automatisch ein Integer oder String sein muss; eben je nachdem, was zuvor für den ersten Parameter als Typ definiert wurde. Sollen stattdessen zwei verschiedene dynamische Typen für beide Parameter zum Einsatz kommen, so hätte man entsprechend auch einen zweiten dynamischen Typ neben *T* für die Funktion definieren müssen; das ist aber nicht der Fall.

Listing 1 zeigt verschiedene Möglichkeiten zum Aufrufen der Funktion mitsamt dem Ergebnis, das der jeweilige Aufruf erzeugt. Dabei zeigt dieses Beispiel sehr schön, wie generisch unsere neue Funktion im Vergleich zu den ersten Beispielen nun ist. Wo anfangs noch für jeden benötigten Typ eine eigene Funktion erstellt wurde, reicht nun eine einzige aus, um alle Anwendungsszenarien erfüllen zu können.

Zu beachten ist dabei lediglich noch, dass in diesem konkreten Beispiel die übergebenen Parameter als Variablen definiert sein müssen, da nur dann deren Werte wie in der Generic Function vorgesehen geändert werden können.

Generics können aber nicht nur für Funktionen zum Einsatz kommen. Es ist auch möglich, eigene sogenannte Generic Types für Klassen, Structures und Enumerations zu definieren. Der Sinn und Zweck und die Funktionsweise von Generic Types ist dabei identisch mit der von Generic Functions, nur mit dem Unterschied, dass ein so definierter dynamischer Typ eben nicht nur innerhalb einer einzigen Funktion, sondern stattdessen innerhalb einer ganzen Klasse, Structure oder eben Enumeration verwendet werden kann.

Listing 1: Aufruf einer Generic Function

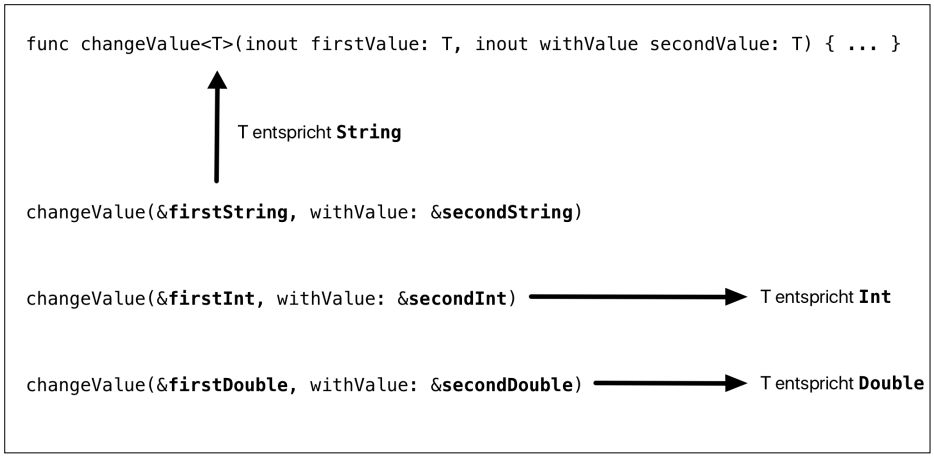
```
var firstName = "Thomas"
var lastName = "Sillmann"
changeValue(&firstName, withValue: &lastName)
print("firstName: \(firstName)")
print("lastName: \(lastName)")
// firstName: Sillmann
// lastName: Thomas

var firstInt = 19
var secondInt = 99
changeValue(&firstInt, withValue: &secondInt)
print("firstInt: \(firstInt)")
print("secondInt: \(secondInt)")
// firstInt: 99
// secondInt: 19
```

Generic Types

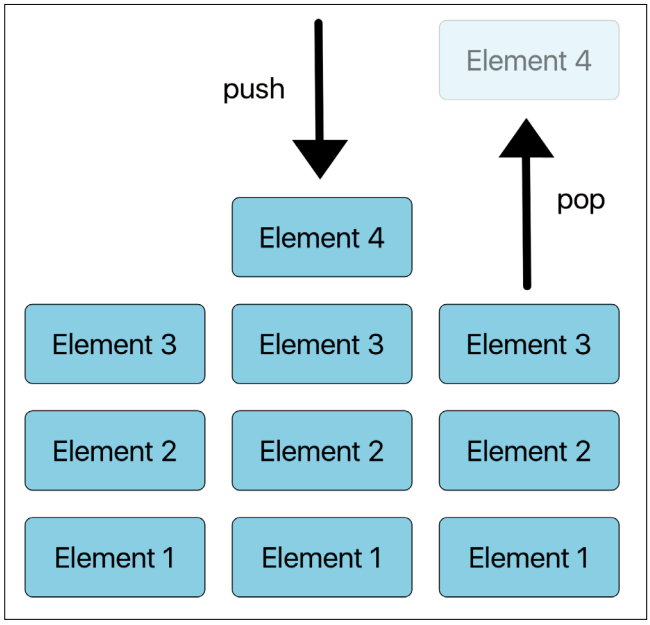
Die Deklaration eines Generic Types ist dabei ganz ähnlich zur Deklaration einer Generic Function. Nach Deklaration der Klasse, Structure oder Enumeration, die über einen Generic Type verfügen soll, werden nach dem Namen innerhalb eines Kleiner- und Größerzeichens die gewünschten Namen für die zu verwendenden Generic Types definiert. Ein solcher Generic Type kann dann innerhalb der Klasse, Structure oder Enumeration verwendet werden wie jeder andere Typ auch. Es können damit also unter anderem Properties oder Parameter für Methoden deklariert werden.

Apple selbst liefert dazu ein sehr schönes Beispiel zur Verdeutlichung von Generic Types, nämlich eine sogenannte Stack-Structure. Diese soll ganz ähnlich funktionieren wie ein Array: Sie hält mehrere verschiedene Objekte eines Typs. Der Unterschied zum Array besteht aber darin, dass neue Objekte immer auf den Stapel bestehender Objekte aufgesetzt werden und beim Entfernen eines Objekts immer das Objekt an oberster Stelle dieses Stapels entfernt wird. Bild 3 verdeutlicht dieses Prinzip.

Um die Funktionsweise und den Vorteil von Generic Types an die-



Übergebener Typ: Je nachdem, welcher Typ beim Aufruf der Funktion übergeben wird, ändert sich der Typ von *T* (Bild 2)



Einem Stack neu hinzugefügte Elemente landen an oberster Stelle und werden auch als Erste wieder entfernt (Bild 3)

ser Stelle zu verdeutlichen, zeigt das folgende Beispiel zunächst einmal eine solche Stack-Structure rein für Strings:

```
struct StringStack {
  var strings = [String]()
  mutating func pushString(string: String) {
    strings.append(string)
  }
  mutating func popString() -> String {
    return strings.removeLast()
  }
}
```

Die Structure *StringStack* ist möglichst einfach gehalten. Sie besitzt ein String-Array, in dem alle String-Objekte des Stapels gespeichert werden, und zwei Methoden, um ein neues String-Objekt dem Stapel hinzuzufügen *(pushString(_:))* und um das letzte Objekt des Stapels wieder zu entfernen *(popString())*. Bei letzterer Methode wird noch das dadurch entfernte String-Objekt des Arrays zurückgeliefert.

Hier ergibt sich nun aber eine ganz ähnliche Problematik wie bei unserer Methode zum Vertauschen der Werte von zwei Variablen. Die erstellte Structure *StringStack* funktioniert ausschließlich mit Strings. Wollen wir eine gleiche Logik für Integer, Double oder andere Typen abbilden, müssten wir entsprechend weitere passende Structures wie *IntStack, DoubleStack* und so weiter schreiben.

Stattdessen wäre es natürlich viel sinnvoller, den gewünschten Typ innerhalb des Stacks wie String oder Integer dynamisch zu ermitteln, je nachdem, wie eine Instanz dieser Structure erstellt wird. Und genau das ist dank Generic Types auch möglich. Dazu wird ein eigener Typ (ich nenne ihn wieder *T)* definiert, der beim Erstellen einer Instanz der Structure dann durch den gewünschten Typ geändert wird, der im ▶

Zusammenhang mit dem Stack verwendet werden soll. Der folgende Code zeigt die entsprechende Implementierung einer solchen Stack-Structure:

```
struct Stack<T> {
  var elements = [T]()
  mutating func pushElement(element: T) {
    elements.append(element)
  }
  mutating func popElement() -> T {
    return elements.removeLast()
  }
}
```

Wieder fällt auf, dass die Implementierung quasi komplett identisch ist mit der aus dem vorigen Beispiel, lediglich mit dem Unterschied, dass überall, wo in *StringStack* der statische Typ *String* zum Einsatz kam, nun der eigens definierte Generic Type *T* verwendet wird. Mit dieser neuen Stack-Structure ist es nun möglich, einen Stack mit jedem beliebigen Typ zu erstellen. Bleibt nur die Frage: Wie teilt man Swift mit, welchen Typ man für den Generic Type eines neuen Stack-Objekts einsetzen und verwenden möchte? Schließlich kann das hier nicht indirekt über Parameter wie bei Funktionen umgesetzt werden.

Tatsächlich ist die Lösung dafür sehr simpel. Genau so, wie der Generic Type einer Klasse, Structure oder Enumeration deklariert wird, wird er auch beim Erstellen einer Instanz eines solchen Elements gesetzt, nämlich nach dem Namen der entsprechenden Klasse, Structure oder Enumeration (in unserem Beispiel Stack) innerhalb eines Kleiner- und Größerzeichens. Das folgende Listing zeigt einige entsprechende Beispiele zum Erstellen einer neuen Stack-Instanz:

```
let stringStack = Stack<String>()
let integerStack = Stack<Int>()
let doubleStack = Stack<Double>()
```

Je nachdem, welcher Typ innerhalb des Kleiner- und Größerzeichens definiert ist, setzt der Stapel dann eben auf entsprechende Objekte jenes Typs.

Type Constraints

So schön die Freiheit und die Flexibilität auch ist, die sich auf die gezeigte Weise mittels Generics erreichen lässt, so sind manchmal trotzdem gewisse Vorgaben nötig, die selbst für die dynamisch definierten Typen gelten sollen, sei es nun, dass ein solcher Typ von einer bestimmten Klasse erbt, oder dass er konform zu einem Protokoll ist.

Glücklicherweise bietet Swift auch hierfür eine Lösung. Mittels der sogenannten Type Constraints ist es möglich, eine Superklasse oder ein Protokoll für einen Generic Type festzulegen, von dem dieser erben beziehungsweise zu dem er konform sein soll.

Die dafür notwendige Syntax ist dabei nichts Besonderes. Im Gegenteil werden Type Constraints für Generic Types auf

Listing 2: Drivable-Protokoll und dazu konforme Structures

```
protocol Drivable {
  var currentSpeed: Int { get set }
  var description: String { get }
  mutating func startDriving()
  mutating func stopDriving()
}
struct Car: Drivable {
  var currentSpeed: Int
  var description: String {
    return "Car"
  }
  mutating func startDriving() {
    print("Car starts driving.")
    currentSpeed = 30
  }
  mutating func stopDriving() {
    print("Car stops driving.")
    currentSpeed = 0
  }
}
struct Motorboat: Drivable {
  var currentSpeed: Int
  var description: String {
    return "Motorboat"
  }
  mutating func startDriving() {
    print("Motorboat starts driving.")
    currentSpeed = 10
  }
  mutating func stopDriving() {
    print("Motorboat stops driving.")
    currentSpeed = 0
  }
}
struct Plane: Drivable {
  var currentSpeed: Int
  var description: String {
    return "Plane"
  }
  mutating func startDriving() {
    print("Plane starts flying.")
    currentSpeed = 100
  }
  mutating func stopDriving() {
    print("Plane stops flying.")
    currentSpeed = 0
  }
}
```

die gleiche Art und Weise definiert wie bei allen anderen Typen auch. Mittels eines Doppelpunkts nach der Definition des Typs folgt kommasepariert die gewünschte Superklasse beziehungsweise die gewünschten Protokolle, zu denen der Generic Type konform sein soll. Das entspricht genau dem Vorgehen, wie es beispielsweise auch bei der Deklaration von Klassen angewendet wird.

Die Auflistung der Klasse und Protokolle, von denen ein Generic Type erbt beziehungsweise zu denen er konform sein soll, erfolgt dabei innerhalb des Kleiner- und Größerzeichens, in dem er definiert wird.

Ein konkretes Beispiel soll den Einsatzzweck und den Sinn von Type Constraints einmal ein wenig veranschaulichen. Dazu wird zunächst ein neues Protokoll namens *Drivable* erstellt, das bestimmte Grundeigenschaften und -funktionen für alle möglichen Arten von Fahrzeugen definiert. Anschließend werden auf Basis dieses Protokolls drei verschiedene Structures definiert, die zum Drivable-Protokoll konform sind und entsprechend alle Eigenschaften und Funktionen implementieren. Diese Grundlage zeigt Listing 2.

Der Einsatz von Type Constraints erfolgt im nächsten Schritt, innerhalb einer neu definierten Structure namens *Garage*. Diese soll alle möglichen Arten von Fahrzeugen in einem Array halten und verschiedene mehr oder weniger sinnvolle Funktionen zum Umgang mit all diesen Fahrzeugen bereitstellen. Dazu erhält *Garage* einen Generic Type namens *Vehicle*, der innerhalb der Structure zum Umgang mit den Fahrzeugen verwendet wird. Ein solcher Generic Type allein würde aber nicht reichen, um eine vernünftige Arbeit mit den verschiedenen Fahrzeugen zu gewährleisten. Damit das funktioniert, müssen sie auch zwingend konform zum Drivable-Protokoll sein. Nur dann können sie auf die gewünschte Art und Weise verwendet werden. Entsprechend wird dem Generic Type *Vehicle* bei dessen Definition in der Structure *Garage* das Protokoll *Drivable* zugewiesen. Dadurch ist Voraussetzung für alle Instanzen der Structure *Garage*, dass ihnen ein Generic Type zugewiesen wird, der konform zum Drivable-Protokoll ist. Listing 3 zeigt eine entsprechend mögliche Implementierung der eben beschriebenen Structure *Garage*.

Theoretisch gibt es auch andere Möglichkeiten, das beschriebene Programmierproblem zu lösen. Es verdeutlich jedoch sehr anschaulich die Funktionsweise von Type Constraints im Zusammenhang mit Generic Types.

Generics und Protokolle

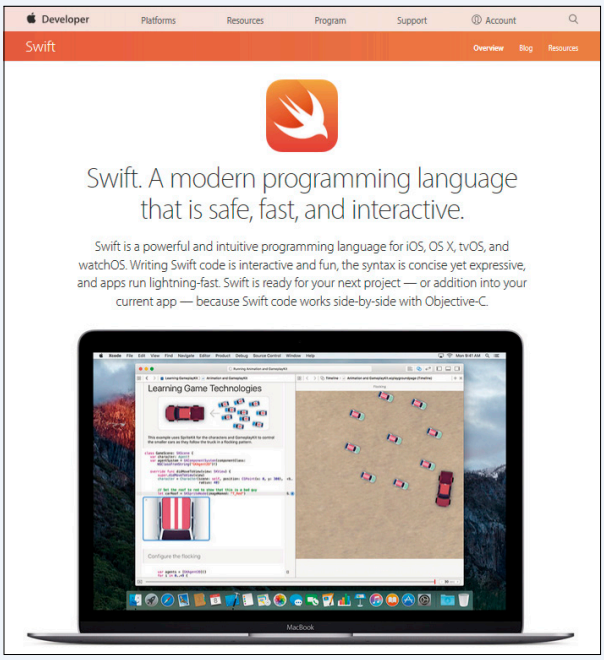
Zum Abschluss dieses Artikels möchte ich einen letzten möglichen Einsatzzweck von Generics vorstellen, nämlich im Zusammenspiel mit Protokollen. Bei der Deklaration von Protokollen arbeiten wir schließlich zwangsläufig auch mit den ▶

Listing 3: Structure Garage mit Generic Type Vehicle

```
struct Garage<Vehicle: Drivable> {
  var vehicles = [Vehicle]()
  func startAllVehicles() {
    for var vehicle in vehicles {
      vehicle.startDriving()
    }
  }

  func stopAllVehicles() {
    for var vehicle in vehicles {
      vehicle.stopDriving()
    }
  }
  func printAllCurrentSpeeds() {
    for vehicle in vehicles {
      print("\(vehicle.description):
      \(vehicle.currentSpeed)")
    }
  }
}
```

Der Ursprung von Swift

Chris Lattner ist Apple-Entwickler und unter anderem für das von ihm initiierte Open-Source-Projekt LLVM (Low Level Virtual Machine) bekannt. LLVM fungiert seit 2008 als Compiler-Unterbau für Apples Xcode. An der Entwicklung von Swift begann Lattner im Juli 2010 aus eigenem Antrieb heraus und zuerst im Alleingang. Gegen Ende 2011 kamen weitere Entwickler dazu. Die Entwicklung fand – wie so oft bei Apple – im Geheimen statt. Lattner benennt insbesondere Objective-C, Rust, Haskell, Ruby, Python, C# und CLU als Einflüsse für seine Programmiersprache. Der Öffentlichkeit vorgestellt wurde die neue Sprache von Apple auf der WWDC 2014.



Swift wurde auf der WWDC 2014 zum ersten Mal der Öffentlichkeit präsentiert

verschiedensten Typen, sei es als Properties oder als Methodenparameter beziehungsweise -rückgabewerte.

Was aber kann man tun, wenn beispielsweise eine Methode eines Protokolls einen Parameter erhalten soll, dessen Typ dynamisch von der jeweiligen Implementierung des Protokolls abhängt? Nehmen wir zur Verdeutlichung noch einmal das Stack-Beispiel, das zu Beginn dieses Artikels vorgestellt wurde. Nehmen wir an, wir möchten diese Klasse über ein eigens definiertes Protokoll erweitern, das die folgenden Eigenschaften definiert:

- Eine Methode, die uns ein Objekt unseres Stacks zu einem bestimmten Index zurückliefert.
- Eine Methode, die unserem Stack ein neues Objekt hinzufügt.
- Eine Methode, die ein Objekt aus dem Stack zu einem bestimmten Index entfernt und das entfernte Objekt zurückliefert.

Jede dieser Eigenschaften benötigt eine essenzielle Information: Welchem Typ entsprechen die Objekte, mit denen interagiert werden soll? Unsere eingangs erstellte Stack-Structure weiß das; dank ihres Generic Types wird ihr dynamisch ein statischer Typ zugewiesen, der dann überall dort, wo zuvor der Generic Type definiert wurde, eingesetzt wird.

Doch nun soll ein davon unabhängiges Protokoll zum Einsatz kommen, das eben diese Information ebenfalls benötigt. Wie kann es diese erhalten?

Um ein besseres Verständnis von der Problematik und der damit einhergehenden Lösung zu erhalten, zeigt das folgende Beispiel einmal eine mögliche Umsetzung des beschriebenen Protokolls (ich nenne es der Einfachheit halber schlicht *StringStackContainer)*, das rein für String-Objekte funktioniert:

Listing 4: Stack-Structure mit Protokoll-Konformität

```
struct Stack<T>: StackContainer {
  var elements = [T]()
  mutating func pushElement(element: T) {
    elements.append(element)
  }
  mutating func popElement() -> T {
    return elements.removeLast()
  }

  // StackContainer Protocol Implementation
  typealias Element = T
  func elementAtIndex(index: Int) -> Element {
    return elements[index]
  }
  mutating func addElement(element: Element) {
    pushElement(element)
  }
  mutating func removeElementAtIndex(index: Int)
  -> Element {
    return elements.removeAtIndex(index)
  }
}
```

```
protocol StringStackContainer {
  func stringAtIndex(index: Int) -> String
  mutating func addString(string: String)
  mutating func removeStringAtIndex(index: Int) ->
  String
}
```

Dieses Protokoll erfüllt seinen Zweck, ist aber auf String-Objekte beschränkt. Unsere Stack-Structure selbst allerdings, die zu solch einem Protokoll konform sein soll, enthält nicht zwingenderweise Instanzen vom Typ *String*. Da sie mittels eines Generic Type definiert ist, kann sie stattdessen alle möglichen Arten von Objekten enthalten. Das Protokoll muss also in der Lage sein, diesen zugehörigen Typ der Stack-Structure ebenfalls dynamisch abzubilden.

Mittels den sogenannten Associated Types ist genau das in Swift auch möglich. Ein Protokoll kann über einen oder mehrere dieser Associated Types verfügen, die lediglich einen generischen Typ-Namen definieren; ganz genau so, wie wir es bisher beispielsweise mit der Bezeichnung *T* oder *Vehicle* gesehen haben. Dieser Associated Type kann dann in der Deklaration des Protokolls auf die gleiche Art und Weise verwendet werden wie jeder andere verfügbare Typ auch.

Doch wie erfolgt die Zuordnung des richtigen statischen Typs, wenn das Protokoll implementiert wird? Dazu wird einfach mittels eines Type Alias in Swift genau diesem generischen Typ der eindeutige Typ zugewiesen.

So weit die Theorie. Für das bessere Verständnis soll das beschriebene Beispiel nun einmal entsprechend umgesetzt werden. Beginnen wir dabei zunächst mit dem Protokoll. Das folgende Beispiel zeigt das Protokoll zur Definition verschiedener Eigenschaften für einen Stack, dieses Mal aber ohne statische Typzuweisung. Stattdessen wird innerhalb des Protokolls ein eigener Associated Type definiert, der dann anstelle des zuvor verwendeten statischen Typs String zum Einsatz kommt:

```
protocol StackContainer {
  associatedtype Element
  func elementAtIndex(index: Int) -> Element
  mutating func addElement(element: Element)
  mutating func removeElementAtIndex(index: Int) ->
  Element
}
```

Wirklich viel hat sich dabei an unserem Code nicht geändert. Gänzlich neu ist lediglich die Deklaration von *associatedtype* Element, das einen generischen Typ namens *Element* für das Protokoll *StackContainer* definiert. Dieser Typ wird dann anstelle von *String* für die übrigen Definitionen innerhalb des Protokolls verwendet. Für *Element* muss dann an der Stelle, an der das Protokoll implementiert wird, der gewünschte korrekte Typ zugewiesen werden.

Listing 5: Verzicht auf explizites Type Alias

```
struct Stack<T>: StackContainer {
  var elements = [T]()
  mutating func pushElement(element: T) {
    elements.append(element)
  }
  mutating func popElement() -> T {
    return elements.removeLast()
  }
  func elementAtIndex(index: Int) -> T {
    return elements[index]
  }
  mutating func addElement(element: T) {
    pushElement(element)
  }
  mutating func removeElementAtIndex(index:Int)-> T {
    return elements.removeAtIndex(index)
  }
}
```

Diesen zweiten Schritt sehen wir uns nun genauer an. Dazu greifen wir auf unsere Stack-Structure zurück und verändern sie so, dass sie konform zum neuen StackContainer-Protokoll ist (Listing 4). Neu ist alles ab dem Kommentar *Stack-Container Protocol Implementation*. Dort findet sich eine passende Implementierung für alle Eigenschaften des Stack-Container-Protokolls. Entscheidend ist dabei die darauffolgende Zeile. Mittels *typealias Element = T* legen wir fest, dass der Associated Type *Element* des StackContainer-Protokolls in dieser Implementierung innerhalb der Stack-Structure dem Typ *T* entspricht; dem von uns definierten Generic Type. Dadurch können wir den Typ *Element* nun, genauso wie im Protokoll definiert, für die drei verschiedenen Methoden verwenden, ohne die Dynamik des Generic Type zu verlieren. Ganz gleich ob es sich dabei dann um einen String, einen Int oder etwas ganz anderes handelt, wird *Element* eben jener Typ zugewiesen und dann entsprechend in den Protokoll-Methoden verwendet. Das Ganze geht noch einen Schritt weiter. Wir können uns nämlich den Befehl *typealias Element = T* in der Implementierung der Stack-Structure sparen und dafür in den drei Protokoll-Methoden anstelle von *Element* unseren gewünschten Typ *T* definieren (Listing 5).

Links zum Thema

- Offizielle Swift-Website
  https://swift.org
- Apple-Dokumentation zu Generics
  https://developer.apple.com/library/ios/documentation/Swift/Conceptual/Swift_Programming_Language/Generics.html

Swift ist so intelligent und erkennt, dass der Associated Type *Element* des Protokolls in dieser Implementierung *T* entspricht. Das muss so sein, schließlich wird *T* an den entsprechenden Stellen verwendet. Durch diese kleine Änderung wird die Implementierung der Stack-Structure deutlich übersichtlicher und wir sparen uns das zusätzliche Type Alias.

Diese Funktionsweise kann dabei äquivalent zu der von Generic Functions gesehen werden. Dort erkennt Swift auch automatisch, welchen Typ es für bestimmte Parameter verwenden soll, abhängig davon, welchem Typ die Objekte entsprechen, die der Generic Function als Parameter übergeben werden. Auf dieselbe Art und Weise arbeitet Swift auch in diesem Fall der Associated Types von Protokollen.

Fazit

Aufgrund ihrer Vielseitigkeit und Dynamik ermöglichen es Generics, deutlich generischeren und damit leichter wiederverwendbaren Code zu schreiben. Die Definition eines generischen Typs, der erst bei Verwendung der zugrunde liegenden Methode, der Klasse, Structure oder Enumeration beziehungsweise des Protokolls zum Einsatz kommt, erlaubt es, Funktionen umzusetzen, die unabhängig vom letztendlichen Typ immer über dieselbe Implementierung verfügen. Wo immer also immer wieder derselbe Code für unterschiedliche Typen definiert und implementiert wird, können stattdessen besser Generics verwendet werden, um den eigenen Code übersichtlicher und schlanker zu halten.

Dabei ist jedoch im gleichen Atemzug daran zu denken, dass Generics Grenzen haben. Abseits der beschriebenen Szenarien ergeben sie nicht wirklich Sinn oder können gar nicht verwendet werden. Sie sind eben – wie es der Name bereits impliziert – absolut generisch. Und was nicht generisch ist, ist auch kein Einsatzzweck von Generics. Lediglich mit Hilfe von Superklassen oder Protokollen können Generics noch minimal angepasst und mit Voraussetzungen verknüpft werden, das war's dann aber auch.

Wer schon immer das Gefühl hatte, des Öfteren denselben Code geschrieben zu haben, nur um mit der immer gleichen Aktion verschiedene Typen zu bedienen, sollte sich Generics auf jeden Fall einmal näher ansehen und daran denken, wenn man in einem Projekt auf eine derartige Situation stößt. Auch wenn die Syntax am Anfang noch etwas gewöhnungsbedürftig sein mag, so geht sie binnen kürzester Zeit in Fleisch und Blut über. Und die Vorteile, die Generics letztlich bieten, sind es allemal wert, sich näher mit der Materie auseinanderzusetzen. ■

Thomas Sillmann
ist iOS-App-Entwickler, Trainer und Autor. Freiberuflich tätig programmiert er für den App Store eigene Apps sowie Apps in Form von Kundenaufträgen. Er ist Autor eines erfolgreichen Fachbuchs und mehrerer Artikel in Fachzeitschriften.
www.thomassillmann.de

Bild: Shutterstock / Alex Mit

CONTAINER-VIRTUALISIERUNG MIT DOCKER

Ab in den Container

Isolierte Container mit Docker erlauben individuelle Entwicklungsumgebungen und erleichtern damit das Deployment.

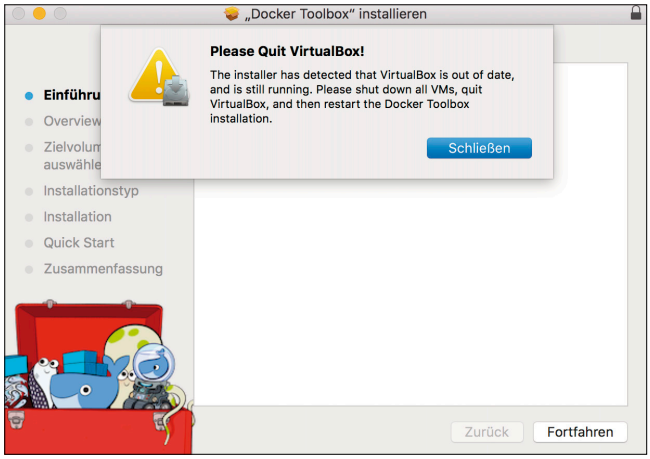
Ob nun virtualisierter Server, Windows-Betriebssystem auf dem Mac oder ein Ubuntu in der VirtualBox – als Entwickler kommt man früher oder später mit Virtualisierung in Kontakt. Das in der Programmiersprache Go geschriebene Docker ist ebenfalls ein Produkt zur Virtualisierung, nämlich zur sogenannten Container-Virtualisierung.

Um das Prinzip zu verstehen, soll kurz Virtualisierung im Allgemeinen vorgestellt werden. Systeme, die virtualisieren, werden als Hypervisor bezeichnet. Hypervisoren haben als Aufgabe die Abstraktion einer Schicht und werden abhängig vom Grad der Virtualisierung in unterschiedliche Kategorien eingeteilt. Typ-1-Hypervisoren abstrahieren direkt die Hardware, das heißt, der Hypervisor muss auch entsprechende Hardwaretreiber für die abstrahierte Hardware bereitstellen. Man spricht hier auch von Hardware-Virtualisierung. Beispiele für Typ-1-Hypervisoren sind KVM oder Xen. Typ-2-Hypervisoren setzen dagegen auf ein Betriebssystem auf und nutzen unter anderem dessen Gerätetreiber (Bild 1). Beispiele für Typ-2-Hypervisoren sind Parallels, VirtualBox oder VMware. In diesem Fall spricht man von Betriebssystem-Virtualisierung.

Betriebssystem	Typ-2-Hypervisor
Typ-1-Hypervisor	Betriebssystem
Hardware	Hardware

Typ-1- und Typ-2-Hypervisor im Vergleich (Bild 1)

Systeme zur Container-Virtualisierung sind grundsätzlich Typ-2-Hypervisoren. Im Ge-



Die Installation der Docker Toolbox erfolgt per Klick-Installer (Bild 2)

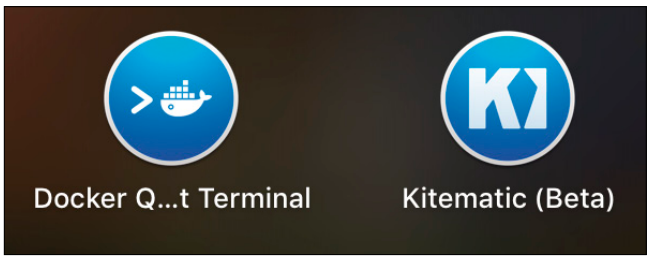
gensatz zu Systemen, die das ganze Betriebssystem virtualisieren, wird bei Container-Virtualisierung der Linux-Kernel untereinander geteilt.

Neben der im Vergleich zu einem klassischen Typ-2-Hypervisor größeren Startgeschwindigkeit eines Containers kommt noch die enorme Einsparung beim Speicherplatz hinzu, denn das Image eines Containers kommt nicht mit einem komplett eigenen Betriebssystem.

Das Image von Ubuntu 15.10 für Docker hat eine Größe von 136,5 MByte (Mac OS X), während das Image für die VirtualBox eine Größe von 4,59 GByte hat. Diesen erheblichen Unterschied spürt man selbst in Zeiten, in denen Speicherplatz en masse verfügbar ist.

Tabelle 1: Bestandteile der Docker Toolbox

Docker CLI	CLI ist die Abkürzung von Command Line Interface. Damit ist es möglich, auf den Docker-Daemon (Docker-Server) mittels REST API zuzugreifen.
Docker Machine	Docker kann nativ nur auf Linux-Systemen laufen, denn es werden die Kernel-Eigenschaften von Linux benötigt. Für den Start auf Mac OS X und Windows wird die Docker Machine benötigt. Diese startet auf Mac OS X und Windows eine virtuelle Maschine mit einem Docker-Daemon, für die Virtualisierung wird das mitgelieferte VirtualBox genutzt. Mit der Docker Machine können auch mehrere Docker-Hosts verwaltet werden.
Docker Compose	Mittels Docker Compose können Architekturen bestehend aus mehreren Docker-Containern definiert und verwaltet werden.
Kitematic	Grafische Oberfläche zur Nutzung von Docker.
Vorkonfiguriertes Shell-Skript	Konfiguriertes Skript für Docker CLI.
Oracle VM VirtualBox	Virtuelle Maschine zum Start von Docker auf Mac OS X und Windows.



Nach der Installation unter Mac OS X stehen das Docker CLI und Kitematic im Launchpad als Icons zur Verfügung (Bild 3)

Docker steht für die Betriebssysteme Linux, Mac OS X und Windows zur Verfügung. Für die Installation unter Linux kann entweder der entsprechende Paketmanager genutzt werden oder ein manueller Download mittels CURL. Für die gängigen Linux-Distributionen steht ein Paket zur Verfügung.

Installation von Docker

Im Fall einer Linux-Installation ist das Installationssystem auch gleichzeitig der Docker-Host, denn der Linux-Kernel muss nicht durch eine zusätzliche Schicht bereitgestellt werden. Das heißt, der Docker-Daemon läuft unter Ausnutzung der Linux-Kernel-Eigenschaften direkt auf dem Linux-System. Die Adressierung eines Docker-Containers erfolgt in diesem Fall über *localhost* und die Nutzung von Ports,

Im Fall von Mac OS X und Windows steht kein Linux-Kernel zur Verfügung und deswegen kann der Docker-Daemon auch nicht direkt auf dem jeweiligen System genutzt werden. Deshalb muss für die Installation unter Windows oder Mac OS X die sogenannte Docker Toolbox installiert werden. Die Docker Toolbox besteht aus mehreren Tools, die in Tabelle 1 kurz erläutert sind. Diese Tabelle illustriert auch das Zusammenspiel der einzelnen Werkzeuge. Die meisten der aufgeführten Werkzeuge stehen auch in der nativen Linux-Version zur Verfügung.

Für die Installation unter Mac OS X muss mindestens ein Mac OS X in der Version 10.8 installiert sein. Für Windows benötigen Sie mindestens Windows 7. Zuerst muss die Docker Toolbox heruntergeladen werden. Die Docker Toolbox kommt mit Oracle VM VirtualBox. Sollte bereits eine VirtualBox auf dem System installiert sein, so muss diese für die ▶

Alternativen zu Docker

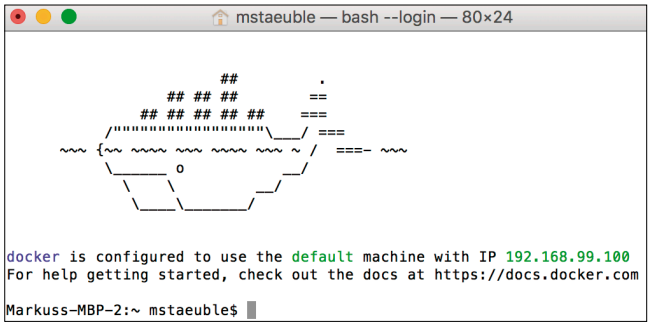
Neben Docker existieren auch etliche andere Systeme zur Container-Virtualisierung, wie zum Beispiel OpenVZ und VServer. Ein sehr interessantes System ist rkt von CoreOS, hier wird besonderer Fokus auf die Sicherheit gelegt. Neben den angebotenen Features ist ein Hauptunterschied zwischen den einzelnen Systemen der genutzte Linux-Kernel. Es gibt Systeme (zum Beispiel Docker) die einen unveränderten Kernel, einen sogenannten Vanilla-Kernel, nutzen und es gibt Systeme, die einen gepatchten Kernel (zum Beispiel OpenVZ) nutzen.

Installation geschlossen werden (Bild 2). Die Installation gestaltet sich ansonsten sehr einfach. Nach wenigen Minuten kann Docker bereits genutzt werden.

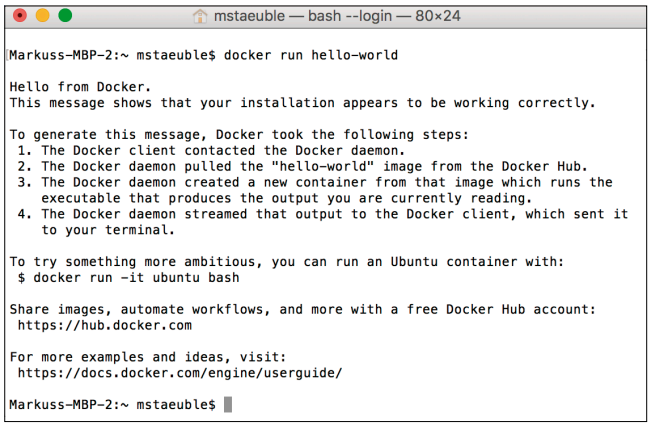
Nach dem Klick auf *Docker Quickstart Terminal* startet in einem neuen Kommandozeilenfenster das Docker CLI (Bild 3). Beim ersten Start dauert es etwas länger. Insbesondere die Erzeugung des SSH-Keys nimmt einige Zeit in Anspruch. Nach dessen Erzeugung startet Docker beim nächsten Mal aber sehr schnell. Sobald Docker erstmalig initialisiert ist, kann das CLI genutzt werden (Bild 4).

Container werden aus einem Image gestartet

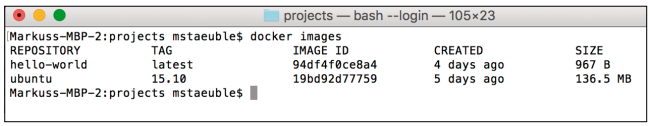
Über das Kommando *docker run hello-world* wird aus dem Docker-Image *hello-world* heraus ein Container gestartet, in dem das Image ausgeführt wird. Sollte das Image lokal noch nicht verfügbar sein, so wird dieses aus einer Registry, dem sogenannten Docker Hub, heruntergeladen. Sollte die zuvor durchgeführte Docker-Installation funktionieren, so erscheint



Per Klick kann Docker nach der Installation gestartet werden (Bild 4)



Der Eingabeprompt signalisiert die Bereitschaft von Docker (Bild 5)



Das hello-world-Image gibt lediglich ein paar Zeilen Text aus (Bild 6)

Kleines Docker-Glossar

Neue Frameworks und Tools bringen auch häufig eine neue Begriffswelt mit sich – so auch bei Docker. Wenn von Docker gesprochen wird, dann wird häufig die sogenannte Docker Engine damit gemeint. Die Docker Engine ist eine Client-Server-Anwendung, bestehend aus dem Docker-Server (Docker-Daemon), dem Docker-Client (Docker CLI) und einem REST-API zur Kommunikation zwischen Client und Server. Neben der Docker Engine gibt es noch die Docker Machine. Diese kann eingesetzt werden, um eine virtuelle Umgebung mit einem Docker-Daemon zu starten, oder auch für die Verwaltung von Docker-Hosts. Ein Docker-Host ist ein System, auf dem eine Docker Engine läuft. Der Docker-Daemon startet aus einem Docker-Image einen Docker-Container. Ein Docker-Image ist ein nicht beschreibbares Template, das alles für den Ablauf der entsprechenden Anwendung enthält. Diese Docker-Images werden in Docker-Registries gespeichert, die bekannteste Docker-Registry ist Docker Hub. Hieraus werden neue Images geladen, falls das aufgerufene Image nicht bereits lokal zur Verfügung steht.

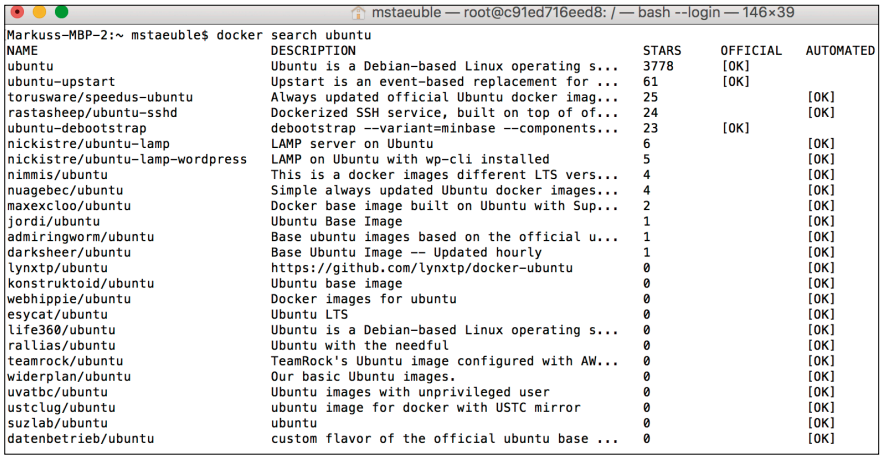
nach Eingabe des Kommandos *docker run hello-world* eine Ausgabe ähnlich der Ausgabe in Bild 5.

Wie beschrieben, wird bei der Ausführung des Befehls *docker run* zunächst geprüft, ob das Image auch lokal verfügbar ist. Um den Download bereits vor der Ausführung durchzuführen, kann über das Kommando *docker pull* ein Image auf das lokale System heruntergeladen werden. Dies verhindert lange Startzeiten beim ersten Start des Containers.

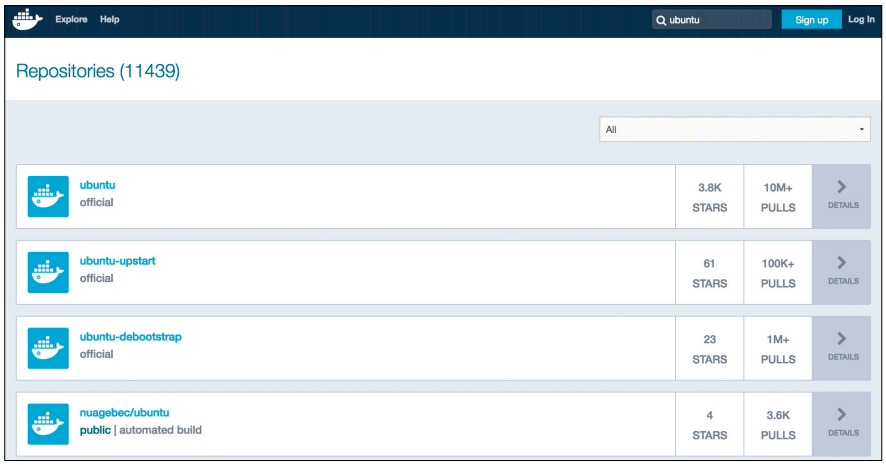
Besonders bei größeren Images ist dies von Vorteil, wie zum Beispiel bei Ubuntu 15.10 mit einer Größe von 136,5 MByte. Um zuerst Ubuntu in Version 15.10 herunterzuladen, muss folgender Befehl abgesetzt werden: *docker pull ubuntu:15.10*. Um zu prüfen, welche Images installiert sind, muss der Befehl *docker images* abgesetzt werden. Es werden dann alle installierten Images aufgelistet.

Jedes Image hat in der Auflistung fünf Attribute. In der Spalte *REPOSITORY* wird der Name des Repositorys aufgeführt. In der Spalte *TAG* wird die installierte Version aufgeführt, denn ein Image kann in mehreren Versionen vorliegen. Diese muss beim Start mit *docker run* angegeben werden. Beim zuvor ausgeführten Kommando *docker run hello-world* wurde auf die Version verzichtet. In diesem Fall wird direkt die aktuellste Version genommen.

Diese Version wird durch *latest* gekennzeichnet, das heißt, der Aufruf ist gleichbedeutend mit *docker run hello-world:latest*. Die *IMAGE ID* ist der eindeutige Bezeichner für das Image. Auch damit kann ein Container gestartet werden. Das in Bild 6 gezeigte Image *hello-world* kann somit auch über *docker run 94df4f0ce8a4* gestartet werden. Die Spalte *CREATED* gibt an, wann das Docker-Image erzeugt wurde. Die Spalte *SIZE* gibt Auskunft über die Größe des installierten Docker-Images. Da man nicht immer den Namen eines Docker-Images kennen kann, gibt es auch Möglichkeiten zur

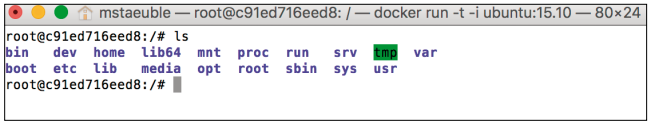


Jedes installierte Image hat eine eindeutige ID (Bild 7)



Alle gefundenen Respositories mit dem Namen *ubuntu* (Bild 8)

Suche. Zunächst sei hier die Suche über die Kommandozeile genannt. Über *docker search* wird im Docker Hub gesucht. Zum Beispiel wird mit *docker search ubuntu* nach *ubuntu* gesucht (Bild 7). Wichtig ist, dass hier nicht nach Images gesucht wird, sondern nach Repositories. Ein Repository enthält Images in unterschiedlichen Versionen. Alternativ zur Kommandozeile kann im Docker Hub auch über den Browser gesucht werden (Bild 8).



Die Suche über die Oberfläche liefert die gleichen Ergebnisse wie die Kommandozeile (Bild 9)

Ein installiertes Image auf dem lokalen System kann man auch wieder löschen. Hierfür steht der Befehl *docker rmi* zur Verfügung. Als Parameter muss der Name oder die ID des Images angeben werden. Um nun das Image *hello-world* zu löschen, muss in diesem Fall der Befehl *docker rmi 94df4f0ce8a4* oder *docker rmi hello-world* abgesetzt werden.

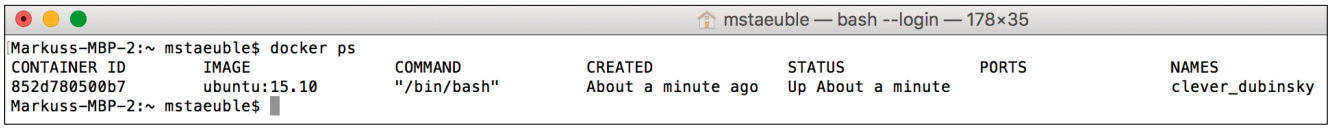
Falls sich das Image nicht löschen lässt, so kann das Löschen mit dem Zusatzparameter *-f* (Abkürzung für *force)* erzwungen werden: *docker rmi -f hello-world*. Bei der Nutzung des Namens versucht Docker, die neueste Version eines Images zu löschen, diese wird durch *:latest* gekennzeichnet.

Das heißt, in diesem Fall wird das Kommando *docker rmi -f hello-world:latest* ausgeführt und das Image wird automatisch gelöscht. Wie in Bild 6 zu sehen ist, ist auf dem System auch noch das Image *ubuntu* in der Version 15.10 installiert.

Ein Aufruf von *docker rmi -f ubuntu* scheitert jedoch, denn dieser Befehl versucht, *ubuntu:latest* zu löschen und dies ist nicht installiert. Entweder wird hier nun die Version angehängt, also *docker rmi -f ubuntu:15.10,* oder es wird über die ID gearbeitet: *docker rmi -f 19bd92d77759.*

Container über die Kommandozeile steuern

Das heruntergeladene Docker-Image *ubuntu:15.10* kann über *docker run* gestartet werden. Beim Start von *docker run ubuntu:15.10* passiert jedoch zunächst nichts. Es muss ein Eingabeterminal angefordert werden (Parameter *-t)* und am besten gibt man noch ein Startprogramm an und wechselt in den interaktiven Modus (Parameter: *-i)*. Der Befehl lautet dann wie folgt: *docker run -t -i ubuntu:15.10 /bin/bash*. Im gestarteten Container kann nun Ubuntu genutzt werden. Dass der Container gestartet ist, erkennt man am geänderten Verzeichnisbaum und auch am geänderten Eingabeprompt (Bild 9). Im gerade ausgeführten Beispiel wird am Eingabeprompt der Benutzer *(root)* und auch die eindeutige ID des Containers *(c91ed716eed8)* angezeigt. Alle laufenden Container kann man sich über den Befehl *docker ps* anzeigen lassen (Bild 10). ▶



Der Container hat seine eigene Verzeichnishierarchie (Bild 10)

Die ID des Containers wird in der der Spalte *CONTAINER ID* dargestellt. In der Spalte *NAME* wird der Name des jeweiligen Containers angezeigt. Der Name kann beim Start über den Parameter *--name* festgelegt werden. Der vergebene Name muss dabei eindeutig sein. Sollte es schon einen Container mit dem angegebenen Namen geben, so wird eine Fehlermeldung ausgegeben.

Um sich mit einem laufenden Container zu verbinden, muss der Befehl *docker attach,* gefolgt von der ID oder dem Namen des Containers, aufgerufen werden. In diesem Fall also der Befehl *docker attach 852d780500b7*. Über den Befehl *docker ps* werden bekanntlich alle laufenden Container angezeigt. Um auch die gestoppten Container anzuzeigen, muss der Befehl *docker ps -a* genutzt werden (Bild 11).

Gestoppte Container werden über den Befehl *docker start*, gefolgt von der ID des Containers, gestartet. Mit *docker start fad2f14a0f72* wird einer der gestoppten Container aus Bild 11 gestartet. Alternativ kann auch das Kommando *docker restart*, gefolgt von der ID, genutzt werden. Über *docker ps* kann geprüft werden, ob der Container wirklich gestartet ist. Über *docker attach fad2f14a0f72* wird die Verbindung zum gerade gestarteten Container hergestellt.

Ein Docker-Container schreibt während seiner Ausführung Logdateien. Diese kann man sich über den Befehl *docker logs*, gefolgt von der ID des Containers, anzeigen lassen.

Bei jedem Aufruf von *docker run* wird ein neuer Container erzeugt. Um einen Container zu löschen, muss das Kommando *docker rm*, gefolgt von der ID des Containers, aufgerufen werden. Bevor ein Container gelöscht werden kann, muss dieser mittels *docker stop* oder *docker kill* gestoppt werden.

Container-Daten dauerhaft speichern

Wie schon erwähnt, sind Docker-Images nicht beschreibbare Templates, aus denen ein Docker-Container gestartet wird. Wenn Docker nun einen Container startet, so wird ein zusätzlicher beschreibbarer Layer erzeugt. In diesem Layer läuft die Anwendung. Dadurch wird erreicht, dass das Image nicht verändert wird und man trotzdem Daten im Container schreiben kann. Die Daten bleiben im Container. Sobald der Container aber gelöscht wird, sind auch die Daten weg.

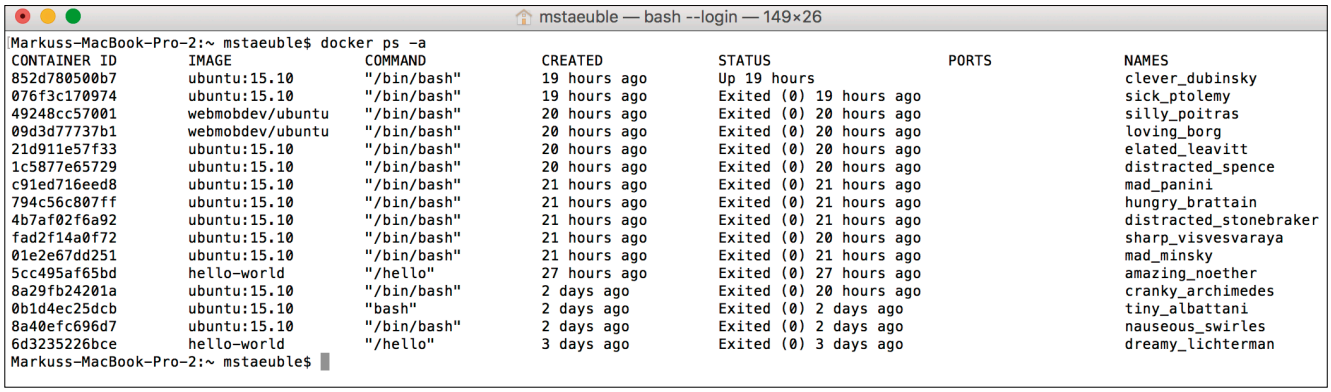
Ein kleines Beispiel soll dies veranschaulichen. Über das Kommando *docker run -i -t --name webmobdev ubuntu:15.10 /bin/bash* wird ein neuer Container mit dem Namen *webmobdev* aus dem Docker-Image *ubuntu* in Version 15.10 gestartet. Das Verzeichnis sieht wie in Listing 1 aus. Nun wird eine neue Datei *webmobdev.txt* mittels Kommando *touch webmobdev.txt* angelegt. Beim Auflisten des Verzeichnisses über das Kommando *ls -ltr* ist diese Datei auch sichtbar (Listing 2).

Listing 1: Verzeichnisbaum des Ubuntu-Containers

```
root@78dbc39a5ce9:/# ls -ltr
total 64
drwxr-xr-x   8 root root 4096 Sep 13  2015 lib
drwxr-xr-x   2 root root 4096 Oct 19  2015 mnt
drwxr-xr-x   2 root root 4096 Oct 19  2015 home
drwxr-xr-x   2 root root 4096 Oct 19  2015 boot
drwxr-xr-x   2 root root 4096 Apr 24 22:19 srv
drwxr-xr-x   2 root root 4096 Apr 24 22:19 opt
drwxr-xr-x   2 root root 4096 Apr 24 22:19 media
drwxr-xr-x   2 root root 4096 Apr 24 22:19 lib64
drwxr-xr-x   5 root root 4096 Apr 24 22:19 run
drwx------   2 root root 4096 Apr 24 22:19 root
drwxr-xr-x   2 root root 4096 Apr 24 22:19 bin
drwxrwxrwt   2 root root 4096 Apr 24 22:19 tmp
drwxr-xr-x  11 root root 4096 Apr 25 17:58 usr
drwxr-xr-x   2 root root 4096 Apr 25 17:58 sbin
drwxr-xr-x  13 root root 4096 Apr 25 17:58 var
dr-xr-xr-x  13 root root    0 Apr 30 10:57 sys
drwxr-xr-x  45 root root 4096 May  2 18:45 etc
dr-xr-xr-x 156 root root    0 May  2 18:45 proc
drwxr-xr-x   5 root root  380 May  2 18:45 dev
```

Listing 2: Verzeichnisbaum des Ubuntu-Containers

```
drwxr-xr-x  45 root root 4096 May  2 18:45 etc
dr-xr-xr-x 156 root root    0 May  2 18:45 proc
drwxr-xr-x   5 root root  380 May  2 18:45 dev
-rw-r--r--   1 root root    0 May  2 18:49
webmobdev.txt
```



Liste der aktiven Container (Bild 11)

Listing 3: Das neu erstellte Image

```
docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
webmobdev_ubuntu    15.10               b4cfefe689c4        32 seconds ago      136.5 MB
hello-world         latest              94df4f0ce8a4        5 days ago          967 B
ubuntu              15.10               19bd92d77759        7 days ago          136.5 MB
```

Listing 4: Ohne Name und Version

```
docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
<none>              <none>              c4cf57135112        2 seconds ago       136.5 MB
webmobdev_ubuntu    15.10               b4cfefe689c4        4 minutes ago       136.5 MB
hello-world         latest              94df4f0ce8a4        5 days ago          967 B
ubuntu              15.10               19bd92d77759        7 days ago          136.5 MB
```

Nach Aufruf von *exit* ist der Container beendet. Mit einem Aufruf von *docker start 78dbc39a5ce9* und *docker attach 78dbc39a5ce9* wird der Container gestartet und man ist dann auch wieder damit verbunden. Ein erneuter Aufruf von *ls –ltr* zeigt, dass die Datei *webmobdev.txt* noch vorhanden ist. Beim Start eines neuen Containers mit *docker run -t -i ubuntu:15.10 /bin/bash* bekommt man das Verzeichnis aus Listing 1 zu sehen, denn der Container wird aus dem Image gestartet, das bekanntlich nicht veränderbar ist. Aus dem laufenden Container kann aber ein neues Image erzeugt werden. Hierfür steht das Kommando *docker commit* zur Verfügung. Dem neu erstellten Image kann ein Name gegeben werden, zum Beispiel *docker commit 78dbc39a5ce9 webmobdev_ubuntu:15.10*. Mit diesem Kommando wird aus dem Container mit der ID *94df4f0ce8a4* ein neues Image erzeugt. Das gespeicherte Image wird beim Aufruf des Kommandos *docker images* in der Liste der auf dem lokalen System vorhandenen Images aufgeführt (Listing 3).

Man braucht beim Kommando *docker commit* nicht zwingend einen Namen für das Image anzugeben. In diesem Fall muss das Image aber immer über die Image-ID referenziert werden. Mit *docker commit 78dbc39a5ce9* wird das Image ohne einen Namen erstellt. Anstelle des Namens steht in den Spalten *NAME* und *TAG* der Wert *<none>* (Listing 4).

Beim Starten eines Containers aus dem neu angelegten Image ist die Datei *webmobdev.txt* auch im Verzeichnisbaum enthalten: *docker run -t -i webmobdev_ubuntu:15.10*

Container sind nicht dafür gedacht, um große Mengen an Anwendungsdaten zu speichern. Diese Daten sollten außerhalb eines Containers liegen. Zum Beispiel sollten die Webseiten einer Webanwendung nicht im Container liegen, damit diese auch von anderen Containern verwendet werden können. Für diesen Zweck gibt es in Docker das Konzept der Data Volumes. Dies sind Verzeichnisse, die außerhalb eines Containers liegen und auch nach dem Löschen des Containers weiterhin existieren. Änderungen in einem Data Volume werden bei einem Absetzen von *docker commit* nicht in das neu erzeugte Image geschrieben.

Um nun ein Verzeichnis vom Host in den Container einzubinden, muss folgender Befehle abgesetzt werden:

```
docker run -t -i -v /Users/mstaeuble/projects/webmobdev-
docker:/webmobdev webmobdev_ubuntu:15.10 /bin/bash
```

Das lokale Verzeichnis */Users/mstaeuble/projects/webmobdev-docker* wird als Verzeichnis *webmobdev* in den Container eingebunden. Änderungen im Verzeichnis innerhalb ▶

Container stoppen und aufräumen

Um eine Containerumgebung zu verlassen, kann man einerseits das Kommandozeilenfenster schließen oder die Tastenkombination [Ctrl]+[p], gefolgt von [Ctrl]+[q], eingeben. Der Container wird dadurch nicht gestoppt, sondern läuft im Hintergrund weiter. Um einen Container zu stoppen, muss der Befehl *exit* eingegeben werden. Alternativ kann man einen Container auch über den Befehl *docker stop*, gefolgt von der ID, anhalten. Sollte der sanfte Stopp nicht funktionieren, so kann über *docker kill*, gefolgt von der ID, der Stopp des Containers erzwungen werden. Wenn ein Container mittels *exit* gestoppt wird, so kann der Container wieder gestartet werden. Um einen Container dauerhaft zu löschen, muss der Befehl *docker rm*, gefolgt von der ID des Containers, ausgeführt werden. Davor muss der Container aber gestoppt werden. Alternativ kann ein Container aber auch nach dem Beenden automatisch gelöscht werden. Hierfür muss beim Start der Parameter *--rm* angegeben werden. Folgender Befehl startet einen Container unter dem Namen *webmobdev* mit dem Image *ubuntu* in Version 15.10 und löscht den Container beim Beenden: *docker run -t -i --rm --name webmobdev ubuntu:15.10 /bin/bash*

des Containers wirken sich direkt auf das lokale Verzeichnis außerhalb des Containers aus.

Beim Einbinden eines Verzeichnisses ist es auch möglich, das eingebundene Verzeichnis nur lesbar einzubinden. Hierfür muss das Flag *:ro* gesetzt werden. Das komplette Kommando lautet wie folgt:

```
docker run -t -i -v /Users/mstaeuble/projects/webmobdev-
docker:/webmobdev:ro webmobdev_ubuntu:15.10 /bin/bash
```

Ein Docker-Image ist ein nicht beschreibbares Template. Auf Basis dieses Templates wird ein Docker-Container gestartet. Ein Docker-Image kann aus mehreren Schichten (Layern) bestehen. Docker nutzt für die Zusammenstellung des Images das UnionFS, das heißt, Dateien und Ordner werden zu einem eigenen Dateisystem zusammengefasst.

Die genutzte Layer-Technologie ist auch ein Grund, warum die Docker-Images so klein sind. Bei der Änderung eines Images wird lediglich ein neuer Layer erzeugt, statt das komplette Image zu kopieren. Bei der Verteilung eines neuen Images muss auch nicht das komplette Image übertragen werden, sondern lediglich die zusätzlich erstellten Layer. Jedes Image nutzt ein sogenanntes Base Image.

Um ein eigenes Docker-Image zu erstellen, gibt es grundsätzlich zwei Ansätze. Der erste Ansatz ist der Einsatz eines vorhandenen Images und die Nutzung des zuvor bereits genutzten Kommandos *docker commit* zur Erstellung eines neuen Images. Dieses Image wird auf Basis eines laufenden Containers, inklusive der darin gemachten Änderungen, erstellt.

Beschreibung eines Images über eine Konfigurationsdatei

Der zweite Ansatz ist die Beschreibung eines Images über eine Konfigurationsdatei, das sogenannte Dockerfile, mit gleichem Namen. Um die Funktionsweise zu zeigen, wird ein kleines Beispiel aus der Docker-Dokumentation erläutert, nämlich ein SSH-Server als Docker-Image. Hierfür wird ein Image auf Basis von Ubuntu in der Version 15.10 erstellt:

```
FROM ubuntu:15.10
```

In diesem Image wird der OpenSSH-Server mittels *apt-get* installiert und das Passwort auf *testpwd* festgelegt. Um einen Container auch von außerhalb zu erreichen, muss ein Port geöffnet werden. Über den Befehl *EXPOSE 22* wird der Port 22 für den Zugriff von außerhalb geöffnet.

Mit dem in Listing 5 gezeigten Dockerfile kann nun das Image erstellt werden. Hierfür muss das Kommando *docker build* genutzt werden. Als Parameter muss der Name des zu erstellenden Images und auch der Ort des Dockerfiles angegeben werden. In diesem Fall lautet das Kommando wie folgt:

```
docker build -t webmobdev-sshd
```

Notwendige Pakete werden anschließend heruntergeladen und das Image mit dem Namen *webmobdev-sshd* wird erstellt. War die Operation erfolgreich, wird dies mit der Anzeige der ID des neuen Images signalisiert.

Mit *docker images* kann geprüft werden, ob das Image auf dem lokalen System vorhanden ist. Nun soll das Image noch getestet werden. Der Container wird mit folgendem Kommando gestartet:

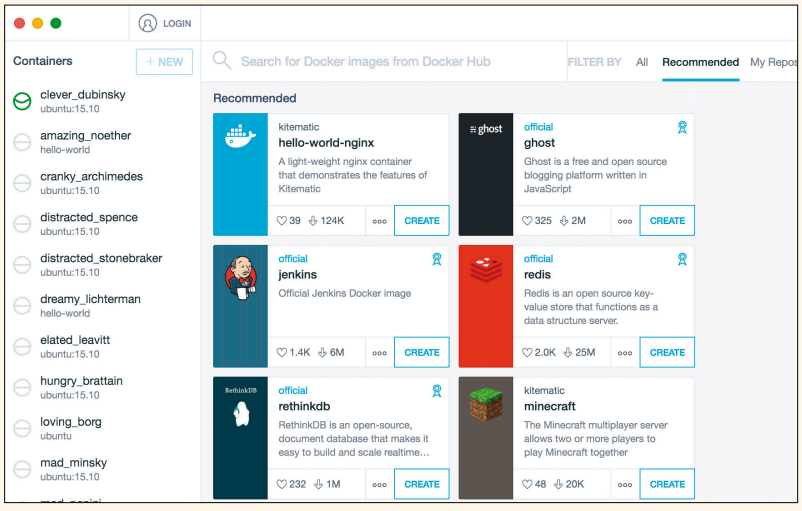
```
docker run -d -P --name webmobdev
webmobdev-sshd
```

In diesem Kommando werden zwei bisher nicht genutzte Parameter verwendet: Über *-d* wird der Container im Hintergrund gestartet und die ID des Containers

Das Docker-GUI Kitematic

Kitematic ist ein grafisches Werkzeug zur Bedienung von Docker. Nach dem Start werden in der linken Spalte alle Container auf dem System aufgelistet. Dabei wird zwischen laufenden und gestoppten Container unterschieden. Die laufenden Container haben ein grünes Icon vor dem Namen des Containers und die gestoppten Container ein graues Icon.

Bei der Selektion eines Containers wird über dem Ausgabefenster für die Logdateien eine Symbolleiste zur Steuerung des Containers angezeigt. Ein laufender Container kann darüber gestoppt (Schaltfläche *STOP)* und auch neu gestartet werden (Schaltfläche *RESTART)*. Ein gestoppter Container hat statt der Startfläche *STOP* die Schaltfläche *START*. Über die Oberfläche von Kitematic kann auch direkt ein neuer Container erstellt werden. Hierfür muss ein entsprechendes Image ausgewählt werden. Die Auswahl ist dabei nicht nur auf die lokalen Images beschränkt, sondern es kann auch im Docker Hub gesucht werden.



Bei der Suche werden Images mit einem Bild und einer Beschreibung angezeigt

wird ausgegeben. Über *-P* werden alle nach außen veröffentlichen Ports an zufällige Ports gebunden. Um eine SSH-Verbindung mit dem SSH-Daemon im Container herzustellen, müssen der Port und die IP-Adresse bekannt sein. Den Port erhält man über das Kommando *docker port webmobdev 22*, die Ausgabe ist ähnlich folgender Ausgabe:

```
docker port webmobdev 22

0.0.0.0:32770
```

Der Port ist in dem Beispiel also 32770. Wenn Docker nicht unter Linux läuft, muss man noch den Port der VirtualBox ermitteln Dies kann man über das Kommando *docker-machine ip* machen, in diesem Fall wird 192.168.99.100 ausgegeben. Nun hat man beide Werte und kann eine Verbindung herstellen. Das Passwort ist das Passwort aus dem Dockerfile.

Links zum Thema

- Docker
  https://www.docker.com
- Docker-Dokumentation
  https://docs.docker.com
- Docker Hub
  https://hub.docker.com
- Docker-Installation unter Linux
  https://docs.docker.com/linux
- Docker-Installation unter Mac OS X
  https://docs.docker.com/mac
- Docker-Installation unter Windows
  https://docs.docker.com/windows
- Docker Toolbox
  https://www.docker.com/products/docker-toolbox
- Go Programming Language
  https://golang.org
- Linux-Kernel-Archiv
  https://www.kernel.org
- LXC
  https://linuxcontainers.org
- OpenSSH
  www.openssh.com
- OpenVZ
  https://openvz.org
- rkt
  https://coreos.com/rkt
- VirtualBox
  https://www.virtualbox.org
- VServer
  http://linux-vserver.org

Listing 5: Dockerfile für SSHD-Image

```
# Version 1.0
# Base-Dockerfile: https://docs.docker.com/engine/
examples/running_ssh_service/

FROM ubuntu:15.10
MAINTAINER none

RUN apt-get update && apt-get install -y
openssh-server
RUN mkdir /var/run/sshd
RUN echo 'root:testpwd' | chpasswd
RUN sed -i 's/PermitRootLogin without-password/
PermitRootLogin yes/' /etc/ssh/sshd_config

RUN sed 's@session\s*required\s*pam_loginuid.
so@session optional pam_loginuid.so@g' -i
/etc/pam.d/sshd

ENV NOTVISIBLE "in users profile"
RUN echo "export VISIBLE=now" >> /etc/profile

EXPOSE 22
CMD ["/usr/sbin/sshd", "-D"]
```

Schnell zu schlanken Umgebungen

Die Startzeit und der geringe Speicherbedarf sind ein klares Argument für die Container-Virtualisierung. Im Vergleich zu anderen Hypervisor-Techniken spürt man hier sofort den Vorteil. Schnell ist eine große Anzahl von Webservern damit gestartet – dies wäre auf einem einzelnen System mit Hardwarevirtualisierung nicht denkbar.

Dazu kommt noch die Möglichkeit, in einer vorhersagbaren und abgeschotteten Umgebung zu entwickeln. Der Container ist jedenfalls schnell wieder neu gestartet. Für Testszenarien ist das eine optimale Voraussetzung.

Auch der Transport der Entwicklungsumgebung wird damit möglich. Oder sei es, um eine lokale Instanz des produktiven WordPress zu starten, um schnell ein Update auszuprobieren. Am Ende wird dann nicht mehr die WordPress-Instanz ausgetauscht, sondern lediglich der Container. Kurzum, Docker ist nicht nur für Administratoren interessant, sondern vor allem auch für Entwickler. ■

Dr. Markus Stäuble
ist passionierter Informatiker, Conference Chair der Developer Week und Programmleiter Make beim Franzis Verlag. Neben Make beschäftigt er sich viel mit dem Thema Mobile und hat zu dessen Auswirkungen auf die Arbeitswelt promoviert.

Foto: a-image / shutterstock.com

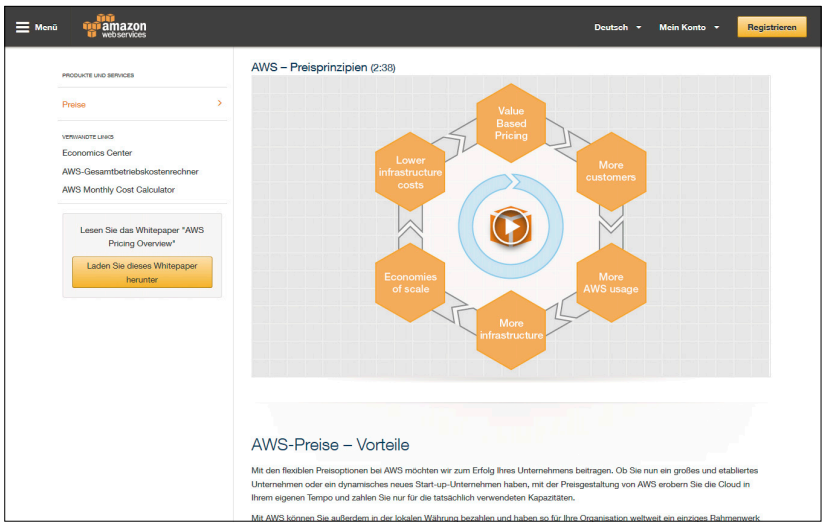
CLOUD-INTEGRATION

Nahtlos verbinden

Hybrid-Cloud-Management-Software holt das Beste aus den beiden Cloud-Welten heraus.

Hybride Cloud-Modelle kommen in Mode. Nach Untersuchungen des Marktforschungsunternehmens Forrester hatten 65 Prozent der für den »Global Business Technographics Infrastructure Survey 2015« Befragten bereits mehr als eine Public- und/oder Private-Cloud-Plattform im Einsatz. In Deutschland ist der Markt noch nicht ganz so weit.

Laut einer Befragung, die das Analystenhaus IDC in deutschen Unternehmen mit mehr als 100 Mitarbeitern durchgeführt hat, nutzten 2015 erst 20 Prozent davon Hybrid-Cloud-Ansätze, ein Jahr zuvor waren es sogar nur 15 Prozent. Immerhin planten schon 57 Prozent den Einsatz.



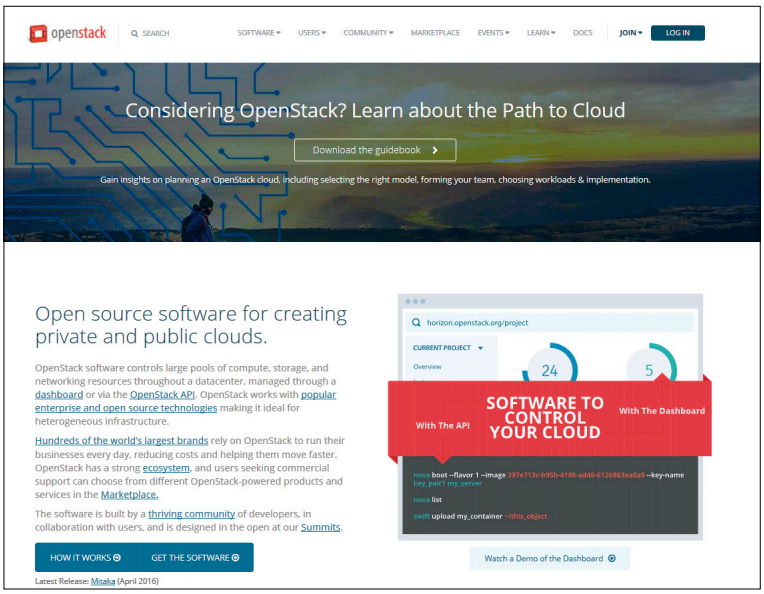
Zu den bekannntesten Cloud-Anbietern zählt Amazon mit seinem AWS (Bild 1)

»Die Nachfrage am Markt bewegt sich definitiv in Richtung hybride Cloud-Plattformen«, sagt Ingo Marienfeld, Geschäftsführer des Softwareherstellers BMC Deutschland. Das Unternehmen verzeichnet laut Marienfeld ein zunehmendes Interesse an seiner Lösung Cloud Lifecycle Management, mit der sich komplexe Cloud-Infrastrukturen aufsetzen und betreiben lassen. Damit spricht Marienfeld eine Herausforderung an, die sich bei hybriden Clouds in besonderer Weise stellt: Wie lassen sich die verschiedenen Cloud-Formen so verwalten, dass keine Reibungsverluste auftreten und wirklich die jeweiligen Vorteile zum Tragen kommen?

Peter Dümig, Senior Server Product Manager bei Dell, das mit Cloud Manager ebenfalls eine Verwaltungsoberfläche für hybride Cloud-Umgebungen anbietet, bestätigt: »Wir sehen einen großen Informationsbedarf nicht nur bei unseren Kunden, sondern auch bei unseren Partnern.«

Wenige konkrete Installationen, aber viel Interesse, konstatiert auch Jörn Kellermann, Senior Vice President Global IT Operations bei T-Systems: »Es gibt kein Kundengespräch, bei dem Hybrid Cloud nicht ein Thema ist.«

Als Schatten-IT sind Hybrid Clouds womöglich verbreiteter als offiziell bekannt, meint Matthias Pfützner, Senior Solution Architect Cloud bei Red Hat: »Viele Unternehmen wissen selbst nicht so genau, ob und wie viele ihrer IT-Lösungen schon in Public-Cloud-Umgebungen laufen.«



Open Source: Zahlreiche Cloud-Lösungen basieren auf OpenStack (Bild 2)

Vorteil Hybrid Cloud

Die Nachfrage nach hybriden Cloud-Modellen kommt laut Forrester vor allem aus dem Software-Bereich. Demnach haben 34 Prozent der für den Developer Survey befragten Entwickler in den vergangenen zwei Jahren bereits Programme geschrieben, die nativ Cloud-Ressourcen einbinden. Viele neue Projekte für hoch skalierbare und flexible Endkundenapplikationen beginnen heute mit einem Prototyp in einer Public Cloud wie Amazon Web Services (Bild 1), Google Cloud Platform oder Microsoft Azure.

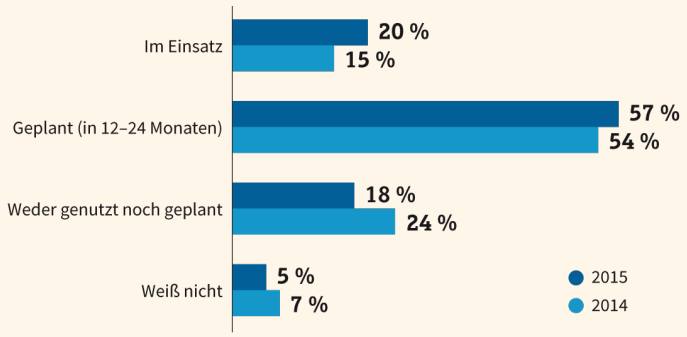
»Entwicklung und Test sind häufig sehr agil und kostengünstig im öffentlichen Bereich der IT-Landschaft durchführbar, solange man anonymisierte Dummy-Daten benutzt«, sagt Ralf Schlenker, Global Practice Lead Cloud and Infrastructure Services beim IT-Dienstleister CGI, der mit seinen Unify360 Hybrid Cloud und IT Services Unternehmen bei der Einführung und dem Management von hybriden Cloud-Modellen berät.

Hybride Plattformen sind auch beliebt, um Lastspitzen abzufangen. »Sollte ein Unternehmenssystem für einen begrenzten Zeitraum zusätzliche Rechen- oder Speicherkapazitäten benötigen, erlaubt es der Hybrid-Cloud-Mechanismus, den privaten Teil der IT für eine begrenzte Zeit zu erweitern«, sagt Yu Wang, Account Director bei ZTE Deutschland.

Der chinesische Netzausrüster hat zur CeBIT eine Lösung vorgestellt, die Private und Public Cloud nahtlos miteinander verbinden kann. Sie basiert auf OpenStack (Bild 2) und richtet sich vor allem an die öffentliche Verwaltung und Großunternehmen. Solche Burst Capacity werde vor allem in Unternehmen gebraucht, deren Bedarf an Rechenleistung stark schwankt, sagt Jörn Kellermann von T-Systems: »Branchen mit Saisongeschäft wie etwa der Handel oder auch die Forschung sind typische Kandidaten für Hybrid Cloud.«

Neben praktischen Aspekten führen aber auch strategische Überlegungen zu einem erhöhten Bedarf an Hybrid-Cloud-Szenarien. Sechs von zehn der von IDC befragten Unternehmen wollen mit dem Aufbau hybrider IT-Landschaften Geschäftsprozesse agiler machen. Jedes fünfte Unternehmen plant, mit Hilfe von Hybrid Clouds neue Produkte, Services und Geschäftsmodelle zu verwirklichen, um so die Digitalisierung voranzutreiben.

Hybrid Cloud: Das Interesse steigt



Im Einsatz: 2014 nutzten erst 15 Prozent der Unternehmen eine Hybrid Cloud, 2015 waren es schon 20 Prozent.

web & mobile developer 8/2016 Quelle: IDC

Herausforderung Integration

Die Kombination verschiedenster Cloud-Quellen miteinander und mit der eigenen, womöglich noch traditionell in Silos aufgebauten IT-Infrastruktur stellt die Verantwortlichen vor große Herausforderungen technischer, organisatorischer und rechtlicher Art. »Es ist keine triviale Aufgabe, hybride Landschaften zu integrieren, zu überblicken, zu kontrollieren, zu steuern und abzusichern«, sagt CGI-Manager Schlenker.

Wer den Weg in die Hybrid Cloud erfolgreich gehen will, brauche profundes Architektur- und Technologiewissen, so T-Systems-Manager Kellermann: »Dieses Know-how fehlt häufig, vor allem in mittelständischen Unternehmen.« Nach Ansicht von Matthias Schorer, ▶

Interview

»Hybrid Cloud wird die Welt verändern«

Roland König, Leiter Geschäftsfeld Virtualisierung der Bechtle AG und Geschäftsführer des Bechtle IT-Systemhauses München/Regensburg, erklärt im Gespräch mit web & mobile DEVELOPER, was Unternehmen bei der Einführung und beim Management von hybriden Cloud-Umgebungen beachten müssen.

Roland König
ist Leiter Geschäftsfeld Virtualisierung der Bechtle AG
www.bechtle.de

web & mobile developer: *Analysten und Berater sprechen und schreiben viel über hybride Cloud-Modelle. In welchem Umfang setzen deutsche Unternehmen Hybrid Cloud überhaupt schon ein?*
Roland König: Viele denken intensiv darüber nach, einige haben bereits damit begonnen, hybride Modelle in die Tat umzusetzen.

web & mobile developer: *Stehen wir also noch ganz am Anfang dieser Entwicklung?*
König: Ja, aber der Markt wächst täglich. Immer mehr Unternehmen nutzen externe Services, das zeigen eindrucksvoll die Steigerungsraten bei den Cloud-Providern. Der Trend zur Digitalisierung, zu Industrie 4.0 zwingt Unternehmen, sich mit der Integration und Vernetzung verschiedenster Komponenten auseinanderzusetzen. Die Frage, ob man in eine Hybrid Cloud gehen soll, stellt sich deshalb meist gar nicht mehr. Es geht eher darum, wann und wie das geschehen soll.

web & mobile developer: *Welche Aufgaben wollen Unternehmen denn in eine Hybrid-Cloud-Plattformen überführen?*
König: Viele möchten beispielsweise das Backup in eine hybride Umgebung verlagern. Auch unabhängige Services, etwa Marketingaktionen, eignen sich gut für den hybriden Ansatz. Mittlerweile betreiben Unternehmen sogar Tier-1-Applikationen wie Exchange komplett in einer hybriden Infrastruktur oder beziehen darüber Services. Office 365 ist dafür ein gutes Beispiel.

web & mobile developer: *Was sind die größten Vorteile einer Hybrid-Cloud-Umgebung?*
König: Es geht vor allem darum, die eigene IT zu entlasten und sie effizienter zu gestalten. Zum zweiten erhoffen sich Unternehmen von einer hybriden Cloud mehr Agilität und Flexibilität für ihre Geschäftsprozesse. Sie wollen auf neue Anforderungen schneller reagieren können.

web & mobile developer: *Und wo liegen Schwierigkeiten?*
König: Die größte Herausforderung ist sicher die interne IT selbst. Sie ist häufig gar nicht darauf vorbereitet, sich in eine hybride Umgebung zu integrieren. Die Strukturen sind teils veraltet, Services und Prozesse nicht klar definiert, und oft fehlen die notwendigen Skills bei den Mitarbeitern. Viele Organisationen sind da noch nicht optimal aufgestellt, um es vorsichtig auszudrücken.

web & mobile developer: *Das heißt, die Unternehmen müssen zunächst intern ihre Hausaufgaben machen.*
König: Manager und IT-Verantwortliche müssen sich heute schon Gedanken machen, wie sie zukünftig externe Ressourcen aus unterschiedlichen Quellen in ihre veränderte, serviceorientierte IT integrieren können. Das ist genauso wichtig, wie einen Service-Katalog zu definieren oder Sicherheitsrichtlinien aufzustellen. Die hybride Cloud muss Bestandteil einer umfassenden IT-Strategie sein.

web & mobile developer: *Was muss man bei der Integration von verschiedenen Cloud-Plattformen und womöglich klassischen internen Rechenzentrums-Ressourcen besonders beachten?*
König: Vor allem muss ich die Betroffenen für das Thema gewinnen und Ängste abbauen. Zudem sollte klar sein, welche Rolle die IT-Abteilung zukünftig spielen wird. Die Kommunikation mit den Fachabteilungen wird deutlich wichtiger, zumal sich die Budgets immer mehr dorthin verlagern. Die IT muss sich als Berater verstehen, der für die Anforderungen aus den Fachabteilungen die bestmögliche Lösung findet, auch wenn diese nicht durch die eigene IT-Fertigungstiefe umgesetzt werden kann. Gerade bei Themen wie Digitalisierung oder Big Data ist das besonders wichtig, da hier sehr häufig hybride Komponenten zum Einsatz kommen. Komplett digitalisierte Prozesse sind ohne Cloud-Integration überhaupt nicht möglich.

web & mobile developer: *Nach welchen Kriterien sollte ein Unternehmen eine Hybrid-Cloud-Management-Lösung auswählen?*
König: Zunächst muss es seine Anforderungen klar definieren. Will das Unternehmen nur unabhängige Services, also einzelne Container nach außen geben oder seine komplette IT nahtlos erweitern? Letzteres funktioniert beispielsweise derzeit nur, wenn intern und extern dieselbe Plattform eingesetzt wird.

Sicherheit spielt natürlich bei allem, was mit Cloud zusammenhängt, eine große Rolle, aber auch die Verwaltung und Automatisierung, um operative Kosten senken und neue Services anbieten zu können. Das entlastet die interne IT von administrativen Aufgaben und schafft zugleich zeitliche Freiräume für neue Projekte.

Dann stellt sich die Frage, wie schnell sich Workloads verschieben lassen. Kann ich virtuelle Maschinen beispielsweise

nicht nur in die Public Cloud migrieren, sondern auch wieder zurückholen? Auch scheinbar banale Fragen wie die nach den Lizenzbedingungen der Hersteller können erhebliche Probleme verursachen. Wenn ich zum Beispiel eine Lizenz dafür habe, eine Plattform als Ganzes zu sichern, kann nicht einfach auf ein containerbasiertes Backup umgestellt werden, dazu brauche ich ein anderes Lizenzmodell.

Am Ende zählen natürlich auch die Abrechnungsmöglichkeiten. Die wenigsten Cloud-Anbieter haben ein transparentes Abrechnungsmodell, das eine einfache Verrechnung im Rahmen einer Multi-Cloud-Lösung erlaubt.

web & mobile developer: *Sollten Unternehmen für das Hybrid-Cloud-Management eher zu unabhängigen Lösungen wie CliQr, RightScale und Scalr oder eher zu den Angeboten der Cloud-Betreiber wie VMware oder HPE greifen?*
König: Man sollte immer darauf achten, wie viel standardisiert ist und wie viel man selbst machen muss. Wir haben viele Beratungsgespräche mit Kunden aus dem OpenStack-Umfeld, die zurück zu Standardprodukten möchten, weil einfach der Aufwand für das Release- und Patch-Management im Open-Source-Umfeld sehr hoch ist. Dies gilt auch für die Integration bestehender Services und Prozesse bei unabhängigen Lösungen. Bei Standardlösungen ist es zudem wesentlich einfacher, Personal mit entsprechenden Kenntnissen zu finden, Mitarbeiter auszubilden und Support zu erhalten als bei Nischenprodukten. Diese mögen an der einen oder anderen Stelle Vorteile haben, aber in der Regel setzen sich doch die Marktführer durch und decken das Angebot zu 90 Prozent ab.

web & mobile developer: *Raten Sie von Lösungen wie CliQr oder Scalr ab?*
König: Nein, das tue ich nicht. Man sollte sich diese Lösungen natürlich ansehen, aber auch berücksichtigen, wie die Wechselmöglichkeiten aussehen, wenn es doch nicht funktioniert. Viele

»Man sollte immer darauf achten, wie viel standardisiert ist und wie viel man selbst machen muss.«

kleine und mittlere Unternehmen werden diese Aufgabe ohnehin eher nach außen geben. Für sie ist es schwierig, die notwendigen Spezialkenntnisse und Mitarbeiter zu finden, die in Entwicklung und Scripting sehr stark sind. Da ist es vielleicht besser, das Hybrid-Cloud-Management als Service einzukaufen. Über den richtigen Weg entscheidet die jeweilige Cloud-Strategie des Unternehmens. Zudem sind Standardprodukte noch sehr teuer. Daher lautet meine Botschaft an die großen Anbieter auch, Cloud-Management-Produkte auf den Markt zu bringen, die für Mittelständler bezahlbar sind.

Head of Strategy Consulting, CEMEA bei VMware, ist vor allem der Datenschutz eine große Aufgabe: »Insbesondere im öffentlichen Bereich und im Gesundheitswesen gibt es strenge Richtlinien.«

Das Cloud-Management muss deshalb genau festlegen, welche Daten in welcher Cloud abgelegt werden dürfen. Schließlich sollen bei Hybrid-Cloud-Projekten auch die Kosten nicht aus dem Ruder laufen. Darüber hinaus erwarten die Nutzer einen nahtlosen Zugang zu allen Ressourcen, ohne sich Gedanken über Richtlinien, Preise oder technische Probleme machen zu müssen.

Was ein HCM können muss

Bei der Integration verschiedenster Cloud-Quellen helfen Systeme für das Hybrid Cloud Management (HCM). Sie bilden nicht nur die Schnittstelle zwischen den verschiedenen Ressourcen, sondern überwachen auch deren Kosten und Leistungsfähigkeit. Nach Ansicht von Forrester-Analyst Dave Bartoletti sollten HCM-Lösungen zwei grundlegende Aufgaben erfüllen:

- Sie sollten die Applikationsentwicklung und -auslieferung erleichtern und beschleunigen. Entwickler können so ohne zusätzlichen Aufwand auf verschiedenste Ressourcen zugreifen und diese miteinander verknüpfen. Das steigert die Produktivität, verkürzt Release-Zyklen und stellt die Konsistenz von Applikationen sicher.
- Sie müssen hybride Infrastrukturen erstellen, optimieren und deren Nutzung lenken können. Eine HCM-Lösung sollte nicht nur Basis-Infrastrukturdienste wie Rechenkapazität, Speicher oder Netzwerkdienste verwalten können, sondern auch Services wie Datenbanken, Messaging, Middleware oder Cloud-Apps. Neben umfangreichen Monitoringfunktionen sind Kostenkontrolle und die Überwachung von Sicherheitsrichtlinien wichtige Merkmale eines guten HCM-Systems. So erhält der IT-Verantwortliche nicht nur eine bessere Sicht auf die hybriden Systeme, sondern kann Ressourcen auch schneller zur Verfügung stellen, Vorgaben durchsetzen und die Kosten im Blick behalten.

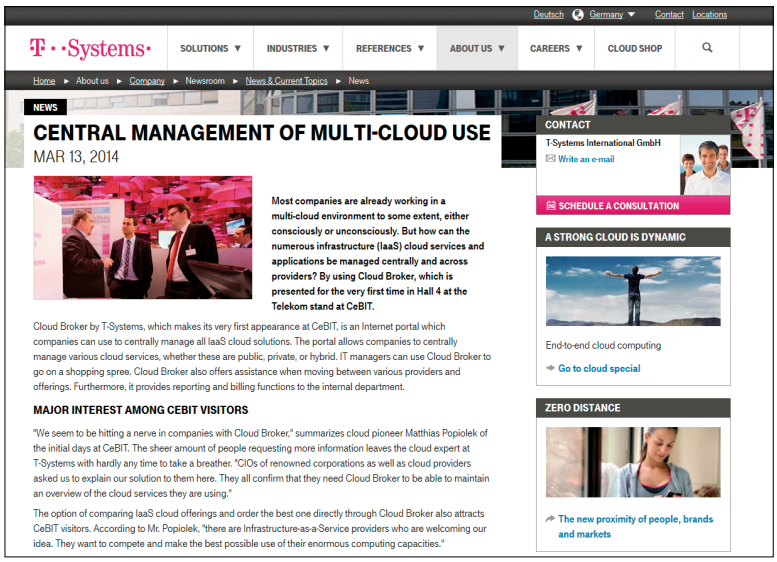
BMC-Manager Marienfeld empfiehlt, für die Auswahl eine Checkliste mit den wichtigsten Funktionen anzulegen: »Wie werden Cloud-Ressourcen angesprochen und eingebunden? Inwieweit lassen sich neue Provider einbinden? Wie sind die Abläufe bei Requests und Fulfillment organisiert? Können Prozesse wie ITSM (IT-Service-Management), CMBD (Configuration Management Database) oder Capacity Planning Out of the Box integriert werden?«

Wenn zusätzlich die Mandantenfähigkeit, ein übersichtliches Monitoring und Reporting sowie Skalierbarkeit gegeben sind, sollten sich Unternehmen näher mit der Lösung befassen, so Marienfeld weiter. Management-Schnittstellen für Deployment, Monitoring und Alerting sowie das Kapazitätsmanagement sind nach Ansicht von VMware-Manager Schorer besonders wichtig: »Es empfiehlt sich, bereits bei der Auswahl des Cloud-Providers auf die Kompatibilität der Schnittstellen zu achten.« ▶

Ralf Schlenker von CGI rät, sich zunächst einmal das eigene Applikations- und Datenportfolio anzusehen und die Anforderungen in den Bereichen Kosten, Leistung, Agilität, Datenschutz, IT-Sicherheit, Architektur und Unternehmensstrategie sorgfältig zu analysieren, um sie verstehen, bewerten und dokumentieren zu können.

»Danach muss eine Gesamtstrategie mit den richtigen hybriden Einzelkomponenten und eine Roadmap – inklusive erster kleinerer Proofs of Concept – entwickelt werden.«

Erst dann sollte sich ein Unternehmen laut Schlenker nach passenden Komponenten und Dienstleistern umsehen. Solch gründliche Vorarbeit hält auch Matthias Pfützner von Red Hat für wichtig: »Entscheidend ist eine Analyse der Anwendungsszenarien und eine an den Geschäftszielen orientierte Implementierung.«



T-Systems hat sich als einer der führenden Cloud-Broker etabliert (Bild 3)

Geteilter Markt

Bei den Angeboten für das Hybrid Cloud Management kann man zwischen zwei prinzipiellen Typen unterscheiden. Dies sind zum einen die Lösungen von Infrastruktur- und Virtualisierungsanbietern wie Hewlett Packard Enterprise (HPE), Citrix, IBM, Microsoft, Red Hat oder VMware.

Sie erweitern meist das eigene Angebot um zusätzliche Public-Cloud-Services. Auf der anderen Seite stehen die spezialisierten Lösungen von Anbietern wie BMC Software, dem gerade von Cisco übernommenen Start-up CliQr, Dell, RightScale oder Scalr, die cloudunabhängig agieren.

CliQr, Dell Cloud Manager, RightScale und Scalr sind nach Ansicht des Forrester-Analysten Bartoletti erste Wahl, wenn es darum geht, Entwickler- und DevOps-Teams zu unterstützen, deren Hauptaufgabe darin besteht, Applikationen über multiple Clouds hinweg zu entwickeln und zu betreiben, und deren Fokus auf Public-Cloud-Angeboten liegt.

Die Lösungen punkten laut Bartoletti mit intuitiv bedienbaren grafischen Bedienoberflächen, mächtigen APIs sowie ausgereiften Funktionen für Template-Erstellung, Automatisierung und Kostenüberwachung. Darüber hinaus unterstützen sie eine große Zahl von Public- und Private-Cloud-Plattformen.

Im zweiten Bereich sind nach Ansicht von Forrester IBM, Red Hat und VMware führend. Diese Lösungen richten sich eher an IT-Verantwortliche, die ihre Private-Cloud-Umgebung mit öffentlichen Clouds erweitern wollen, sei es um Lastspitzen abzufangen oder interne Applikationen funktional zu erweitern.

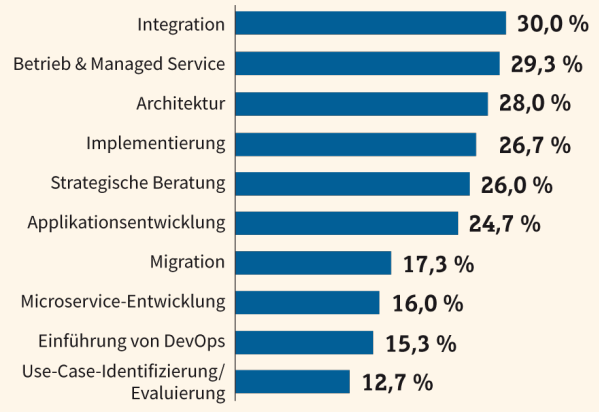
»Diese Plattformen bieten umfassenden Support für vorgefertigte Anwendungen und Infrastruktur-Templates, ein leistungsfähiges Provisionierungs- und Konfigurationsmanagement, eine rollenbasierte Steuerung und vielfältige Funktionen für das Kosten, Leistungs- und Kapazitätsmanagement«, sagt Forrester-Analyst Bartoletti.

Dienstleistung als Alternative

Für viele Unternehmen ist die Verwaltung einer hybriden Cloud selbst mit Hilfe der Management-Suiten eine große Herausforderung. Unterstützung finden sie bei Dienstleistern. Einige Service-Provider sammeln als Hybrid-Cloud-Broker-Dienste verschiedener Cloud-Anbieter auf einer Plattform. Der Markt sei derzeit allerdings noch überschaubar, so die Analysten der Experton Group.

»Die lokalen Umsatzerlöse schätzen wir aktuell noch sehr moderat ein«, heißt es dazu im Cloud Vendor Benchmark Deutschland 2015. Die Experten erwarten allerdings, dass das sich der Markt in Hinblick auf Kundenakzeptanz, Angebot und Anbieter rasch entwickeln wird. Aktuell als führend nennt Experton den Cloud-Broker von T-Systems (Bild 3), das Cloud-Services-Brokerage-Angebot von Capgemini und den Cloud Platform Broker von Accenture.

Was der Mittelstand braucht



n = 150
(Mehrfachnennungen)

In welchen Bereichen der Cloud benötigen Sie Unterstützung?

Cloud-Projekte: Den größten Bedarf an Hilfe sehen mittelständische Unternehmen bei Integration und Betrieb.

web & mobile developer 8/2016 Crisp Research AG

T-Systems punktet laut Experton unter anderem mit einer Transformation Engine, die eine einfache und schnelle Migration zwischen verschiedenen IaaS-Angeboten ermöglicht, Capgemini sorgt für eine sehr gute Vergleichbarkeit der einzelnen Services und Accenture ist vor allem im Bereich Kostenanalyse und Budgetkontrolle erste Wahl.

Services für Hybrid Cloud Management

T-Systems hat im Umfeld von Hybrid Cloud Management noch eine ganze Reihe weiterer Services im Portfolio. So bietet das Cloud Integration Center ähnlich wie die genannten HCM-Lösungen eine einheitliche Managementoberfläche für verschiedene Cloud-Varianten. Das Verwaltungssystem ist allerdings nicht frei auf dem Markt erhältlich, sondern nur für T-Systems- beziehungsweise Telekom-Kunden gedacht. Der Cloud Connector stellt eine Verbindung zwischen internen VMware-Ressourcen und der ebenfalls auf VMware-Technologie basierenden DSI vCloud der Telekom her. Der Cloudifier schließlich erlaubt es, Anwendungen mit relativ überschaubarem Aufwand cloudfähig zu machen. »Cloud ist ja nicht nur ein Infrastrukturthema, sondern geht auch mit ganz anderen Konzepten bei der Software-Architektur einher«, sagt T-Systems-Manager Kellermann.

Der IT-Dienstleister CGI bietet unter der Marke Unify360 ein Portfolio aus Beratung, IT-Integrationsplattform und Management für hybride IT-Umgebungen, das auch Verwaltung und Services für Hybrid-Cloud-Systeme umfasst. Ähnlich wie die Cloudifier von T-Systems sollen außerdem die Application Management Services Kunden dabei helfen, ihre Applikationen auf agile und schnelle Cloud-Modelle umzustellen.

Fazit

Keine Frage: Hybride Cloud-Modelle liegen im Trend. Das ist nachvollziehbar, denn die Kombination aus klassischen IT-Ressourcen, Private-Cloud-Ansätzen und ergänzenden Services aus der Public Cloud bietet die größtmögliche Flexibilität beim Aufbau neuer Applikationen oder Dienstleistungen. Klug kombiniert und sorgfältig gemanagt lassen sich dabei nicht nur alle rechtlichen und regulatorischen Vorgaben einhalten, sondern auch noch ganz erheblich Kosten sparen. Eine gute Hybrid-Cloud-Management-Lösung ist dafür eine wichtige Voraussetzung. Welche man wählt, hängt in erster Linie davon ab, was man mit der Hybrid Cloud bezweckt.

Wer vor allem Workloads zwischen internen Ressourcen und der Public Cloud hin und her schieben will, der ist mit Lösungen von HP, IBM, Microsoft Red Hat oder VMware am besten bedient. Allerdings sollte dann auch die interne IT auf der Virtualisierungsplattform des jeweiligen Anbieters basieren. Unabhängige Cloud-Lifecylce-Management-Systeme wie die von BMC, CliQr, Dell, ScalR oder RightScale sind dagegen vor allem für Anwendungsfälle interessant, in denen es auf größtmögliche Plattformunabhängigkeit und Flexibilität ankommt.

Vor allem mittelständische Unternehmen sollten auch ein Auge auf die Service-Angebote von T-Systems, CGI und Co. werfen. Sie können den Einstieg in ein hybrides Cloud-Modell deutlich erleichtern. ■

Anforderungen an ein HCM

Diese Kriterien sind für die Auswahl einer HCM-Lösung entscheidend:

- Einfache Integration möglichst vieler Plattformen
- Direkte Ressourcen-Bereitstellung für Entwickler und Anwender
- Rollenbasierte Zuweisung von Ressourcen
- Vorlagen für die einfache Erstellung und Integration von neuen virtuellen Maschinen, Servern, Containern oder Applikationen
- Automatische Kostenkontrolle
- Überwachung von Sicherheits- und Compliance-Richtlinien
- Monitoring und Reporting für Verbrauch und Performance

Hybrid-Cloud-Management-Systeme

- BMC Software / Cloud Lifecycle Management
  www.bmcsoftware.de/it-solutions/cloud-lifecycle-management.html
- Citrix / Lifecycle Management
  www.citrix.com/products/citrix-lifecycle-management
- CliQr Technologies / CloudCenter
  www.cliqr.com
- Hewlett Packard Enterprise / HPE Helion CloudSystem
  www8.hp.com/us/en/cloud/cloudsystem.html
- Red Hat / CloudForms
  www.redhat.com/en/technologies/cloud-computing/cloudforms
- RightScale / Cloud Portfolio Management
  www.rightscale.com
- Scalr / Cloud Management Platform
  www.scalr.com
- VMware / vRealize Automation
  https://www.vmware.com/de/products/vrealize-automation
- T-Systems / Cloud Integration Center
  www.t-systems.com/de/de

Dr. Thomas Hafen
ist seit mehr als 15 Jahren als Redakteur und Journalist tätig, unter anderem für die IT-Fach¬zeitschriften NetworkWorld Germany und ChannelPartner sowie die Fotoseiten Seen.by und Digitalkamera.de.
http://thomas-hafen.de

MIT DEM RASPBERRY PI 3 ENTWICKELN

Raspberry Pi to go

Entwickler, die mit dem Raspberry Pi arbeiten möchten, benötigen weder Monitor noch Tastatur am Pi, sondern nur eine funktionierende Remote-Umgebung.

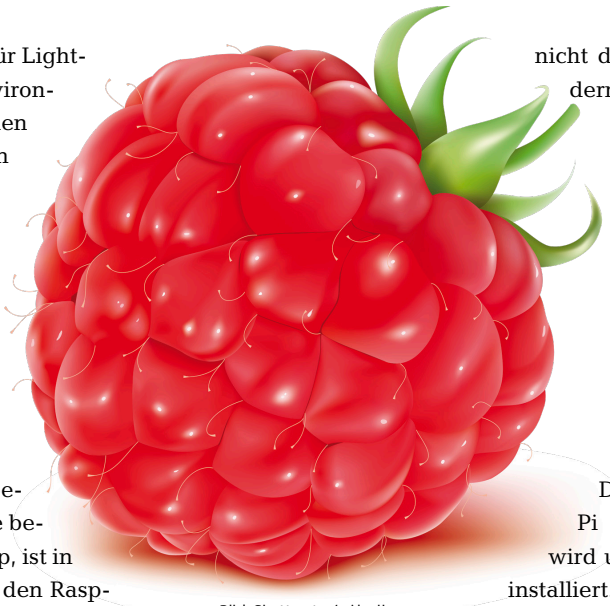


Bild: Shutterstock / ivelly

Klar, mit LXDE (Abkürzung für Lightweight X11 Desktop Environment) bietet der Raspberry Pi einen grafischen Desktop, mit dem man gut arbeiten kann. Ob nun die Office-Suite LibreOffice, der Browser Epiphany oder Minecraft, die Software-Auswahl für den Minicomputer ist groß. Über LXDE lässt sich alles steuern, und mit dem neuesten Raspberry Pi 3 lässt sich damit auch flott arbeiten.

Da man aber als Entwickler bereits eine Entwicklungsmaschine besitzt, ob nun Laptop oder Desktop, ist in der Regel ein Remotezugriff auf den Raspberry Pi komfortabler – dadurch muss man nicht noch eine zusätzliche Tastatur, Maus und Monitor im Zugriff haben.

Der Raspberry Pi wird in diesem Artikel deswegen im Headless-Modus betrieben, das heißt, Sie brauchen lediglich eine Speicherkarte, einen WLAN-Router (oder ein USB-TTL-Kabel) und ein Netzteil, um mit dem Minicomputer arbeiten zu können.

Um ganz auf zusätzliche Peripherie für den Raspberry Pi verzichten zu können, soll auch die Installation des Pi ohne Monitor, Tastatur und Maus erfolgen. Aufgrund dessen wird nicht der Installer NOOBS genutzt, sondern direkt ein Image ausgewählt.

Sie lernen, wie Sie den Raspberry Pi optimal für Entwicklung per SSH, X-Server oder VNC einrichten und Dateien direkt auf dem Raspberry Pi speichern.

Betriebssystem für den Raspberry Pi installieren

Um den Raspberry Pi im Headless-Modus zu betreiben, wird ein minimales Setup benötigt. Als Datenspeicher setzt der Raspberry Pi auf eine MicroSD-Karte, hierauf wird unter anderem das Betriebssystem installiert.

Hier sollte man nicht am falschen Ende sparen und zu einer etwas teureren Class-10-Karte greifen. Für die Stromversorgung wird ein 5-Volt-Netzteil benötigt. Die Raspberry Pi Foundation empfiehlt ein Netzteil, das 2,5 Ampere Strom liefern kann. Für die erste Konfiguration bietet sich noch ein Ethernetkabel an, grundsätzlich kann aber auch darauf verzichtet werden.

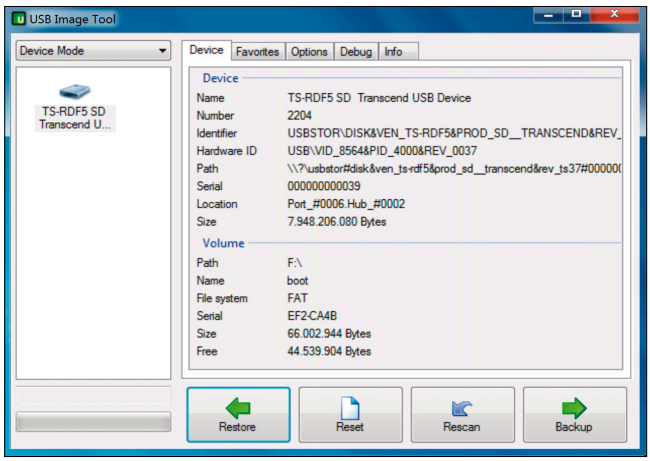
Wie schon erwähnt, wird kein NOOBS für die Installation verwendet, sondern direkt das entsprechende Image heruntergeladen. In diesem Fall wird das aktuellste Raspbian Jessie (2016-05-10) genutzt.

Von dieser Distribution stehen zwei Images, jeweils als ZIP-Archiv, zum Download bereit: einmal eine Variante mit grafischer Oberfläche (Größe des entpackten Images: 4,02 GByte) und einmal ohne grafische Oberfläche (Größe des entpackten Images: 1,39 GByte).

Die Installation beider Images verläuft identisch: Nach dem Entpacken des entsprechenden ZIP-Archivs muss das Image auf die SD-Karte überspielt werden. Unter Windows stehen dazu diverse grafische Tools zur Verfügung, wie zum Beispiel USB Image Tool (Bild 1) oder Win32 Disk Imager.

Unter Linux oder Mac OS X kann die SD-Karte auch direkt mit Bordmitteln über die Kommandozeile beschrieben werden. Dafür muss die SD-Karte zuerst identifiziert werden, dies erledigt man über den Aufruf *diskutil list*. Die externe SD-Karte ist mit *external* markiert (Listing 1).

Im nächsten Schritt muss die SD-Karte zunächst mittels dem Kommando *diskutil unmount /dev/disk2* ausgehängt werden, der Name *disk2* muss dabei auf das jeweilige Sys-



Über die Schaltfläche Restore wird das Image auf die SD-Karte geschrieben (Bild 1)

tem angepasst werden. Im Anschluss daran kann die SD-Karte beschrieben werden:

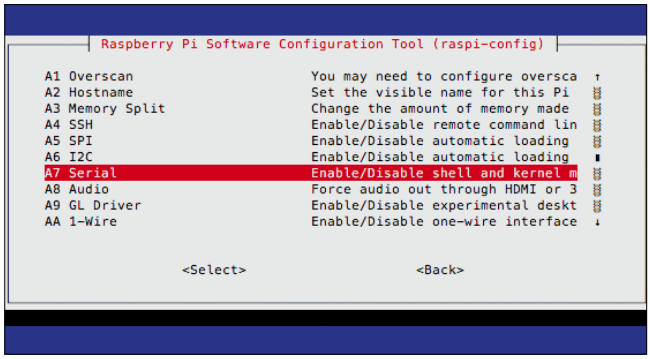
```
sudo dd bs=1m if=2016-05-10-raspbian-jessie
of=/dev/rdisk2.img
```

Das Beschreiben kann einige Zeit in Anspruch nehmen, einen Status erhält man über die Tastenkombination [Ctrl T]. Nach Fertigstellung kann der Raspberry Pi mit der beschriebenen SD-Karte gebootet werden.

Nach der Installation von Raspbian Jessie kann über den Benutzer *pi* und das Passwort *raspberry* auf den frisch installierten Raspberry Pi zugriffen werden. Da dieses Passwort jeder kennt, sollte man über die Eingabe von *passwd* das Passwort nach der ersten Anmeldung direkt ändern.

Zugriff ohne Monitor

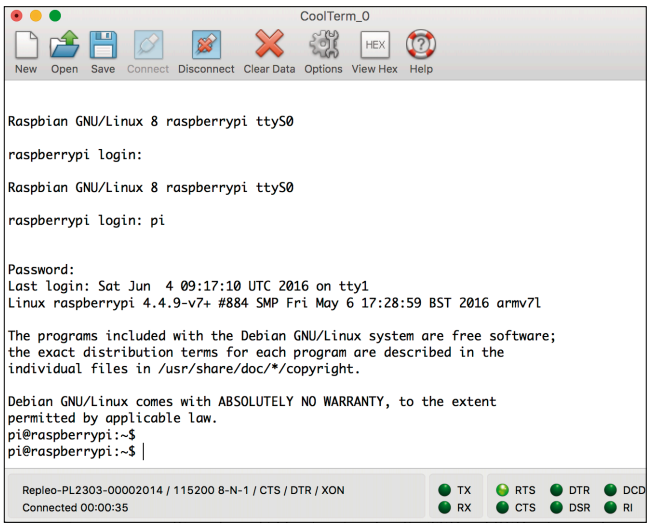
Nach dem Booten kann der Zugriff auf den Raspberry Pi erfolgen. Für den Zugriff über die Konsole gibt es mehrere Möglichkeiten: SSH und per serieller Schnittstelle. Für den Zugriff über die serielle Schnittstelle wird ein USB-TTL-Kabel benötigt. Der Vorteil dieser Variante: Man benötigt nicht unbedingt eine Netzwerkverbindung über ein lokales Netzwerk. Hierfür müssen Sie den seriellen Zugriff aber auch aktivieren. Sollten Sie bereits auf dem Raspberry Pi arbeiten können, so können Sie diesen Zugriff über das Konfigurationsprogramm raspi-config aktivieren (Bild 2).



Aktivierung des seriellen Zugriffs über raspi-config (Bild 2)

Listing 1: Die externe SD-Karte

```
$ diskutil list

/dev/disk0 (internal, physical):
/dev/disk1 (internal, virtual):
/dev/disk2 (external, physical):

   #:  TYPE NAME                 SIZE       IDENTIFIER

   0:  FDisk_partition_scheme    *7.9 GB    disk2
   1:  Windows_FAT_32 boot       66.1 MB    disk2s1
   2:  Linux                     7.9 GB     disk2s2
```



Serielle Verbindung mit dem Raspberry Pi über CoolTerm (Bild 3)

Um ein USB-TTL Kabel auf Linux, Mac OS X oder Windows nutzen zu können, muss ein entsprechender Treiber installiert sein und das Kabel muss mit den richtigen Pins am Raspberry Pi verbunden sein. Bevor Sie direkt an den Raspberry Pi gehen, sollten Sie prüfen, ob die Treiberinstallation auf dem Hostrechner geklappt hat. Hierfür stecken Sie den USB-Stecker in einen freien Port, öffnen ein Terminalprogramm und starten eine serielle Verbindung.

Unter Windows eignet sich hierfür zum Beispiel PuTTY, bei Mac OS X bietet sich CoolTerm oder auch die Kommandozeile an. Jetzt verbinden Sie TXD mit RXD (weißes und grünes Kabel), zum Beispiel über eine Büroklammer. Alles, was Sie nun auf der Tastatur eintippen, sollte direkt im Terminalprogramm ausgegeben werden. Wenn dem so ist, dann funktioniert das Kabel an Ihrem Rechner und Sie können mit dem Raspberry Pi voranschreiten und die vier Kabel mit der GPIO verbinden.

Die richtige Verkabelung

Wichtig ist, dass das schwarze GND-Kabel tatsächlich mit der Masse verbunden ist und nicht mit 5 V, ansonsten können Sie direkt ein neues Kabel kaufen. Eine Masseverbindung ist über mehrere GPIO-Pins möglich, zum Beispiel über den dritten Pin von oben auf der äußeren GPIO-Leiste.

Die Spannungsversorgung des Raspberry Pi sollten Sie extern einrichten und nicht über das rote 5-V-Kabel. Das heißt, dieses Kabel verbinden Sie nicht mit Ihrem Raspberry Pi. Die anderen beiden Kabel müssen mit TXD (weiß) und RXD (grün) verbunden sein. TXD ist der vierte Pin von oben auf der äußeren GPIO-Leiste. RXD ist der fünfte Pin von oben auf der äußeren GPIO-Leiste.

Jetzt können Sie eine serielle Verbindung mit 115.200 Baud öffnen und sollten die Kommandozeile des Raspberry Pi vor sich haben (Bild 3). Wenn das nicht klappt, so tauschen Sie TXD und RXD untereinander aus. Unter Linux und Mac OS X können Sie auch einfach die Kommandozeile nutzen. Hierfür müssen Sie lediglich wissen, wie die serielle Schnittstelle auf Ihrem System heißt. Auf dem genutzten Testsystem war es ▶

/dev/tty.Repleo-PL2303-00002014. Die serielle Verbindung kann in diesem Fall dann über den Kommandozeilenbefehl *screen /dev/tty.Repleo-PL2303-00002014 115200* hergestellt werden.

Netzwerkzugriff

Da nicht jeder solch ein USB-TTL-Kabel besitzt, sollen nun noch die anderen Zugriffsmöglichkeiten gezeigt werden – zuerst der Zugriff per Kommandozeile mittels SSH. Der SSH-Server wird sowohl beim Image mit grafischer Oberfläche gestartet als auch beim Image ohne grafische Oberfläche.

Damit ist ein Zugriff ohne angeschlossenen Monitor direkt möglich. Ohne konfiguriertes WLAN bleibt nur der Zugriff über ein Ethernet-Kabel. Die IP-Adresse des Raspberry Pi ermittelt man am besten über die Oberfläche des genutzten Routers.

Nach der Anmeldung über *pi* und das Passwort *raspberry* hat man Zugriff auf die Konsole. Nach dem Beschreiben des Images sollte man zuerst den gesamten Speicher der SD-Karte dem Raspberry zur Verfügung stellen. Dafür sollte man sich zunächst einen Überblick über den belegten Speicher verschaffen.

Den belegten Speicherplatz kann man sich über den Kommandozeilenbefehl *df –h* anzeigen lassen. In Listing 2 ist zu sehen, dass lediglich 1,2 GByte der Speicherkarte genutzt werden und davon sind bereits 74 Prozent in Gebrauch. Um die komplette Speicherkarte nutzen zu können, muss die vorhandene Linux-Partition vergrößert werden.

Bevor es nun an die Vergrößerung der Partition geht, muss diese zunächst gelöscht werden. Bei der Installation von Raspbian Jessie werden zwei Partitionen angelegt, eine Boot-

Listing 2: Belegter Speicherplatz

```
pi@raspberrypi:~ $ df -h
Filesystem      Size  Used Avail Use% Mounted on
/dev/root       1.2G  816M  290M  74% /
devtmpfs        459M     0  459M   0% /dev
tmpfs           463M     0  463M   0% /dev/shm
tmpfs           463M  6.2M  457M   2% /run
tmpfs           5.0M  4.0K  5.0M   1% /run/lock
tmpfs           463M     0  463M   0% /sys/fs/cgroup
/dev/mmcblk0p1   60M   20M   41M  34% /boot
pi@raspberrypi:~ $
```

Listing 3: Die beiden genutzten Partitionen

```
pi@raspberrypi:~ $ mount | grep ^/dev
/dev/mmcblk0p2 on / type ext4
(rw,noatime,data=ordered)
/dev/mmcblk0p1 on /boot type vfat (rw,relatime,
fmask=0022,dmask=0022,codepage=437,iocharset=ascii,s
hortname=mixed,errors=remount-ro)
```

Listing 4: Anzeige der Partitionen mit fdisk

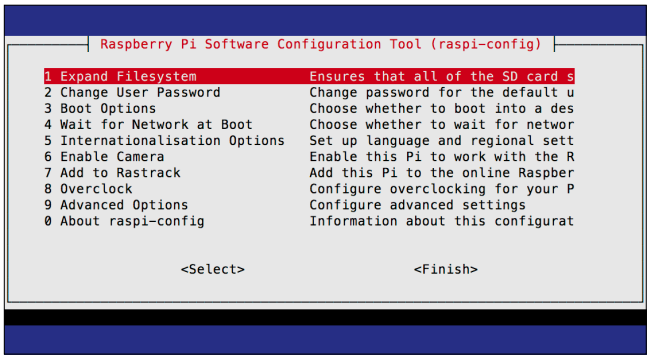
```
pi@raspberrypi:~ $ sudo fdisk /dev/mmcblk0

Welcome to fdisk (util-linux 2.25.2).
Changes will remain in memory only, until you decide
to write them.
Be careful before using the write command.

Command (m for help): p
Disk /dev/mmcblk0: 7.4 GiB, 7948206080 bytes,
15523840 sectors
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512
bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disklabel type: dos
Disk identifier: 0x6f92008e

Device         Boot  Start     End Sectors  Size Id
Type
/dev/mmcblk0p1        8192  131071  122880   60M  c
W95 FAT32 (LBA)
/dev/mmcblk0p2      131072 2658303 2527232  1.2G 83
Linux
```

Partition *(/dev/mmcblk0p1)* mit einem FAT32-Dateisystem und eine Linux-Partition *(/dev/root)* mit EXT4-Dateisystem. Die zwei Dateisysteme kann man sich über den Befehl *mount* in Verbindung mit einem regulären Ausdruck anzeigen lassen (Listing 3).

Die *boot*-Partition ist für das Bootstrapping notwendig und sollte deswegen unverändert bleiben. Die andere Partition kann man aber bedenkenlos ändern. Hierfür wird diese über den Befehl *fdisk* gelöscht und dann vergrößert.

Zuerst wird über das Kommando *sudo fdisk* das Programm *fdisk* gestartet. Über Eingabe von *p* werden die vorhandenen Partitionen angezeigt (Listing 4).

Gelöscht werden soll nun die Linux-Partition. Wichtig zu erwähnen ist, dass beim Löschen einer Partition keine Daten



Über den Menüpunkt Expand Filesystem wird die Partition vergrößert, auch hier ist ein anschließender Reboot nötig (Bild 4)

Listing 5: Anlegen der neuen Partition

```
Command (m for help): n
Partition type
   p   primary (1 primary, 0 extended, 3 free)
   e   extended (container for logical partitions)
Select (default p): p
Partition number (2-4, default 2): 2
First sector (2048-15523839, default 2048): 131072
Last sector, +sectors or +size{K,M,G,T,P} (131072-
15523839, default 15523839):

Created a new partition 2 of type 'Linux' and of
size 7.3 GiB.
```

Listing 6: Die neu angelegte Partition

```
Device         Boot  Start      End  Sectors  Size Id Type
/dev/mmcblk0p1        8192   131071   122880   60M  c W95 FAT32 (LBA)
/dev/mmcblk0p2      131072 15523839 15392768  7.3G 83 Linux
```

Listing 7: Die neuen Größenverhältnisse

```
pi@raspberrypi:~ $ df -h
Filesystem      Size  Used Avail Use% Mounted on
/dev/root       7.2G  817M  6.1G  12% /
devtmpfs        459M     0  459M   0% /dev
tmpfs           463M     0  463M   0% /dev/shm
tmpfs           463M  6.2M  457M   2% /run
tmpfs           5.0M  4.0K  5.0M   1% /run/lock
tmpfs           463M     0  463M   0% /sys/fs/cgroup
/dev/mmcblk0p1   60M   20M   41M  34% /boot
```

verloren gehen. Hierfür muss man sich den Start-Sektor notieren. Über das Kommando *d* und Eingabe von *2* wird die Partition mit Endung *p2* gelöscht. Über das Kommando *n* wird dann eine neue Partition angelegt (Listing 5). Über *p* kann man sich schließlich das Ergebnis ansehen (Listing 6).

Mit dem Kommando *w* wird die Partitionstabelle geschrieben. Um die Änderungen wirksam werden zu lassen, muss ein Reboot über *sudo reboot* durchgeführt werden. Nun muss noch das Filesystem über *sudo resize2fs /dev/mmcblk0p2* angepasst werden. Über *df -h* sieht man dann endlich das Ergebnis (Listing 7).

Alternativ kann auch das Programm raspi-config genutzt werden. Da dieses konsolenbasiert ist, ist eine Nutzung über SSH möglich (Bild 4).

Um im Headless-Modus auf die grafische Oberfläche zugreifen zu können, muss entweder ein X-Server oder VNC genutzt werden. Für die Arbeit mit dem kompletten LXDE-Desktop ist VNC notwendig. Hierfür wird ein VNC-Server

Listing 8: Nummer des Displays

```
pi@raspberrypi:~ $ sudo vncserver

New 'X' desktop is raspberrypi:1

Starting applications specified in /root/.vnc/
xstartup
Log file is /root/.vnc/raspberrypi:1.log
```

auf dem Raspberry Pi benötigt. Wer sich nicht händisch mit den Einstellungen der Bildschirmauflösung beschäftigen möchte, sollte zu TightVNC Server greifen.

Über das Kommando *sudo apt-get update* werden die Paketinformationen in Raspbian auf den aktuellen Stand gebracht. Nun kann der VNC-Server mittels *sudo apt-get install tightvncserver* installiert werden. Über *sudo vncserver* wird der VNC-Server als Hintergrundprozess gestartet. Beim erstmaligen Start muss ein Passwort vergeben werden, dieses wird in *.vnc/passwd* gespeichert. Nach dem Start erscheint die Nummer des Displays in der Konsole (Listing 8).

Für die nun folgende Verbindung ist die Displaynummer wichtig, diese steht hinter dem Rechnernamen. Die angegebene Zahl muss zum VNC-Standardport 5900 addiert werden. In diesem Beispiel ist der Port somit 5901.

Über einen VNC-Client, wie zum Beispiel Ultra VNC oder VNC Viewer, kann der Zugriff auf den Raspberry-Pi-Desktop erfolgen. Beim Aufbau der Verbindung muss das zuvor ▶

Setup für den Artikel

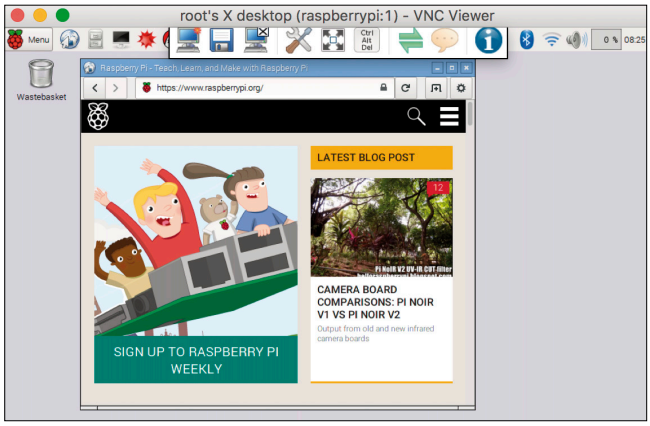
Für den Artikel wurde ein aktueller Raspberry Pi 3 Modell B benutzt. Der Raspberry Pi ist dabei über ein Ethernetkabel an einen sehr günstigen Nano-Router von TP-Link (TL-WR702N) angeschlossen, und ins Internet geht es über eine FRITZ!Box 7390. Der Nano-Router wurde dabei lediglich für die erste Konfiguration genutzt, danach erfolgte der Zugriff über WLAN mittels FRITZ!Box 7390. Für die serielle Verbindung wurde ein USB-TTL-Kabel von Adafruit genutzt. Bei direkter Konfiguration der Datei *wpa_supplicant.conf* in der EXT4-Partition kann auf den anfänglichen Zugriff per Ethernetkabel verzichtet werden. Um unter Windows oder Mac OS X auch EXT4 schreiben zu können, wird ein zusätzliches Tool benötigt, zum Beispiel Paragon ExtFS. Der Hostrechner (Mac OS X 10.11.13) ist dabei über WLAN mit dem Nano-Router verbunden. Als Speicherkarte kommt eine Class-10-Karte mit 8 GByte von SanDisk zum Einsatz. Als Betriebssystem auf dem Raspberry Pi ist Raspbian Jessie mit einem Releasedatum 2016-05-10 im Einsatz. Für das Beschreiben der SD-Karte wurde ein Kartenlesegerät von Transcend, der TS-RDF5K, eingesetzt.

vergebene Passwort angegeben werden. Nach Etablierung der Verbindung wird die grafische Oberfläche angezeigt (Bild 5). Der VNC-Server wird über den Parameter *-kill* und Angabe der Displaynummer wieder gestoppt: *sudo vncserver -kill :1*.

Wenn man nur einzelne grafische Programme nutzen möchte, so ist die Nutzung eines X11-Servers definitiv die bessere Wahl. Auf dem Raspberry Pi muss hierfür nichts Zusätzliches installiert werden. Lediglich eine Konfigurationseinstellung in der Datei *sshd_config* im Verzeichnis */etc/ssh* ist notwendig:

```
X11Forwarding yes
X11DisplayOffset 10
```

Auf dem Rechner, auf dem die Programmfenster angezeigt werden sollen, muss ein X-Server gestartet werden. Sollte kein X-Server installiert werden, so bietet sich Xming für Windows an und XQuartz für Mac OS X. Zusätzlich muss beim Aufruf von *ssh* der Parameter *X* mit angegeben werden.



LXDE mit gestartetem Dateimanager (Bild 5)

Listing 9: Der WLAN-Chip wurde erkannt

```
wlan0     Link encap:Ethernet  HWaddr xx:xx:xx:xx:xx:xx
          inet6 addr: fe80::8969:7bb6:212b:304b/64 Scope:Link
          UP BROADCAST MULTICAST  MTU:1500  Metric:1
          RX packets:556 errors:0 dropped:556 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:219239 (214.1 KiB)  TX bytes:0 (0.0 B)
```

WLAN konfigurieren

Der Raspberry Pi 3 kommt direkt mit einem integrierten WLAN-Chip. Zunächst sollte geprüft werden, ob der Chip auch erkannt wurde. Dies kann man über den Kommandozeilenbefehl *ifconfig -a* erledigen. In der Ausgabe müsste ein Block mit *wlan0* erscheinen (Listing 9).

Damit auch die richtigen Funkkanäle genutzt werden können, muss das richtige Land in den Einstellungen von raspi-config ausgewählt werden. Über *Internationalisation, Options* und *Change Wi-fi Country* lässt sich das Land anpassen (Bild 6). Für Deutschland muss *DE Germany* ausgewählt werden.

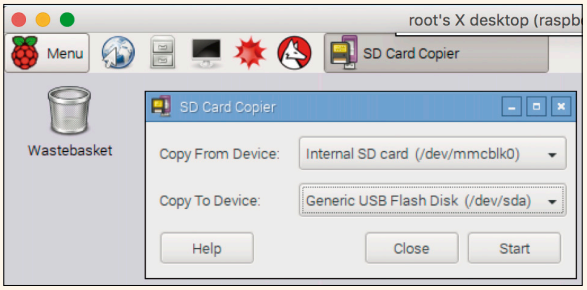
Vor der Herstellung der Verbindung mit dem entsprechenden Netzwerk kann man noch über das Kommando *iwlist wlan0 scan | grep "ESSID"* prüfen, welche WLAN-Netzwerke für den Raspberry Pi in Reichweite sind. Sollte das entsprechende Netzwerk in Reichweite liegen, so muss dieses in */etc/wpa_supplicant/wpa_supplicant.conf* eingetragen werden (Listing 10). Nach dem Speichern der Konfigurationsdatei muss das Netzwerk neu gestartet werden: *sudo service networking restart*. Anschließend sollte der Zugriff mittels WLAN möglich sein.

Software wird beim Raspberry Pi über die von Debian bekannte Paketverwaltung APT (Advanced Packaging Tool) verwaltet. APT nutzt dabei intern dpkg (Abkürzung für Debian Package) – die Pakete haben die Dateiendung *.deb*. APT

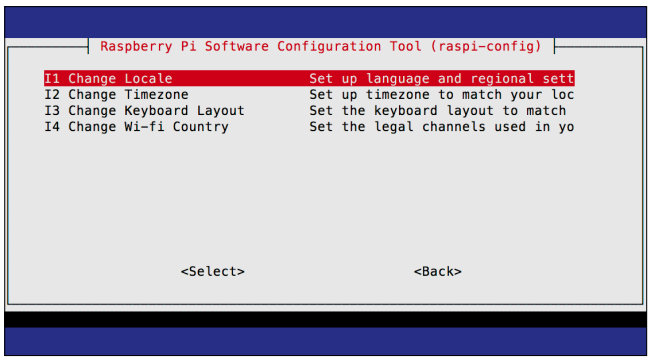
Backup und Migration der SD-Karte

Die SD-Karte enthält alle Daten einer Raspberry-Pi-Installation und sollte deswegen auch gesichert werden. Mit dem letzten Update von Raspbian Jessie (Release date 2016-05-27) kommt ein grafisches Programm hierfür mit: SD Card Copier.

Damit kann die genutzte SD-Karte kopiert werden. Hierfür muss eine zusätzliche SD-Karte über einen USB-SD-Kartenleser im Raspberry Pi gesteckt werden. Diese wird für *Copy to Device:* ausgewählt. Nach einigen Minuten ist die SD-Karte kopiert. Damit ist es möglich, dass die Ziel-SD-Karte größer oder kleiner ist. Falls die Karte kleiner als die zu kopierende Karte ist, so wird gemeldet, dass der Speicher nicht ausreicht. Alternativ kann das Image auch über die Kommandozeile per *dd* erzeugt werden, hierfür muss ebenfalls ein USB-Stick oder Netzwerkspeicher zur Verfügung stehen. Und schließlich kann das Image auch über Linux oder Mac OS X erzeugt werden. Damit die Imagedatei nicht zu groß wird, kann diese direkt bei der Erzeugung komprimiert werden: *sudo dd bs=4M if=/dev/sda | gzip > raspbian.img.gz*



Quelle des Kopiervorgangs ist immer die interne SD-Karte



Um die in Deutschland zugelassenen Kanäle zu nutzen, muss auch Deutschland als *Wi-fi country* gesetzt werden (Bild 6)

Listing 10: SSID und Passwort

```
country=DE
ctrl_interface=DIR=/var/run/wpa_supplicant
GROUP=netdev
update_config=1
network={
  ssid="NETZWERK_NAME"
  psk="PASSWORT"
}
```

besteht aus mehreren Konsolenprogrammen, die beiden wichtigsten sind *apt-get* und *apt-cache*.

APT speichert auf dem lokalen System Paketlisten, diese müssen aktuell gehalten werden. Die eigentlichen Pakete werden aus definierten Repositories geladen. Die zugelassenen Repositories sind in der Datei */etc/apt/sources.list* konfiguriert (Listing 11).

Paketlisten von APT

Wichtig ist es, die Paketlisten von APT auf dem aktuellen Stand zu halten. Dies erledigt man am einfachsten mit *apt-get*. Der Aufruf auf der Kommandozeile dazu lautet *sudo apt-get update*. Jetzt werden die lokalen Paketlisten aktualisiert, eine Aktualisierung von bereits installierten Paketen erfolgt in diesem Schritt jedoch noch nicht.

Mit APT kann ein neues Paket installiert werden. Zusätzlich kann ein vorhandenes Paket aktualisiert und gelöscht werden. Die Installation wurde bereits in obigen Beispielen eingesetzt.

Über den Aufruf *sudo apt-get install* wird ein neues Paket installiert. Hierfür muss der Name des Pakets angegeben werden, zum Beispiel wie bereits zuvor beim TightVNC Server verwendet: *sudo apt-get install tightvncserver*.

Listing 11: Inhalt von /etc/apt/sources.list

```
deb http://mirrordirector.raspbian.org/raspbian/
jessie main contrib non-free rpi
# Uncomment line below then 'apt-get update' to
enable 'apt-get source'
#deb-src http://archive.raspbian.org/raspbian/
jessie main contrib non-free rpi
```

Listing 12: Metadaten

```
pi@raspberrypi:/etc/apt $ apt-cache show
tightvncserver
Package: tightvncserver
Source: tightvnc
Version: 1.3.9-6.5
Architecture: armhf
Maintainer: Ola Lundqvist <opal@debian.org>
Installed-Size: 1383
```

Die Aktualisierung aller installierten Pakete erfolgt über *sudo apt-get upgrade*. Zuvor müssen aber die Paketlisten über *sudo apt-get update* aktualisiert werden. Soll nur ein einzelnes Paket aktualisiert werden, so muss über *sudo apt-get install* und Paketname gearbeitet werden.

Gelöscht wird ein Paket über *sudo apt-get remove,* gefolgt von dem Paketnamen, zum Beispiel *sudo apt-get remove tightvncserver*. Um das Paket komplett mit allen Konfigurationsdateien zu löschen, muss der Befehl *sudo apt-get purge,* gefolgt von dem Paketnamen, in der Kommandozeile abgesetzt werden.

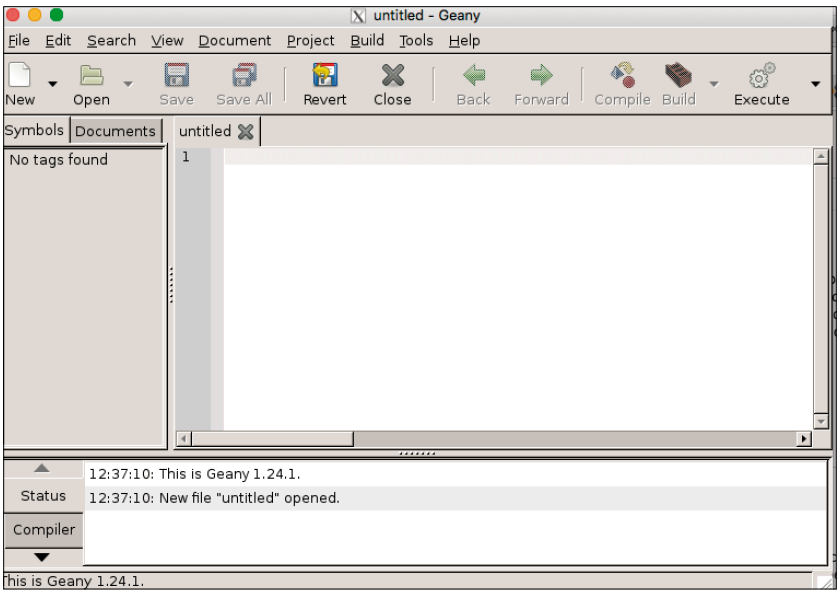
Mit dem Konsolenprogramm *apt-cache* können die Metadaten der Pakete durchforstet werden. Über *apt-cache show,* gefolgt von dem Paketnamen, werden Informationen zu einem installierten Paket ausgegeben (Listing 12). Falls man den Namen eines Pakets nicht genau kennt, so kann man mit *apt-cache search* nach Paket suchen (Listing 13). ▶

Listing 13: Volltextsuche

```
pi@raspberrypi:/etc/apt $ apt-cache search vncserver
libvncclient0 - API to write one's own vnc server - client library
libvncclient0-dbg - debugging symbols for libvncclient
libvncserver-config - API to write one's own vnc server - library utility
libvncserver-dev - API to write one's own vnc server - development files
libvncserver0 - API to write one's own vnc server
libvncserver0-dbg - debugging symbols for libvncserver
tightvncserver - virtual network computing server software
vnc4server - Virtual network computing server software
xtightvncviewer - virtual network computing client software for X
xvnc4viewer - Virtual network computing client software for X
```

Bei Programmierprojekten kommen schnell einige Daten zusammen und hier stellt sich die Frage, wie man am besten die Daten vom Hostrechner auf den Raspberry Pi überträgt. Einzelne Konfigurationsdateien lassen sich noch schnell per *vi* oder *nano* über die SSH-Konsole editieren, doch bei einem größeren Python-Projekt ist eine IDE von Vorteil. Eine gute Möglichkeit ist die Nutzung von VNC und X-Server (Bild 7), dann werden die Daten direkt auf der SD-Karte des Raspberry Pi gespeichert.

Falls die SD-Karte zu klein wird oder die Daten ausgelagert werden sollen, kann auch NFS genutzt werden und ein Verzeichnis des Hostrechners eingebunden werden. Wem der Zugriff über FTP zu mühsam ist, der kann auch einen Samba-Server auf dem Raspberry Pi aufsetzen. Hierfür muss der Samba-Server zuerst über *apt-get* installiert werden:



Geany ist ein schlanker Python-Editor, der in diesem Fall auf dem Hostrechner über X11-Forwarding läuft (Bild 7)

Um auch auf dem lokalen Raspberry-Pi-System die Freigaben zu testen, sollte direkt noch der Samba-Client mittels *sudo apt-get install smbclient* installiert werden. Zunächst sollte geprüft werden, ob die notwendigen Dienste gestartet wurden:

```
service smbd status
service nmbd status
```

Eine Ausgabe von *service smbd status* sollte ähnlich der Ausgabe in Listing 14 sein. Nun muss der Samba-Server noch konfiguriert werden. Die Konfigurationsdatei *smb.conf* liegt im Verzeichnis */etc/samba*. Am besten wird die bestehende Konfigurationsdatei gesichert und eine neue Datei angelegt. In der *smb.conf* werden die über Samba erreichbaren Verzeichnisse freigegeben.

Da der Zugriff über Benutzer und Passwort in Listing 15 konfiguriert ist, muss noch ein Passwort über den Aufruf von *smb-*

upgrade oder dist-upgrade

Neben *apt-get upgrade* kann zur Aktualisierung der installierten Pakete auch *apt-get dist-upgrade* genutzt werden. Auch wenn die beiden Kommandos sehr ähnlich arbeiten, ist es wichtig, die Unterschiede zu kennen. Mit *apt-get upgrade* werden alle auf dem System installierten Pakete aktualisiert. Hierfür werden aus den in */etc/apt/sources.list* aufgeführten Repositories die neuesten Versionen der installierten Paket heruntergeladen und installiert.

Während des Vorgangs werden keine vorhandenen Pakete gelöscht, und es werden auch keine neuen Pakete durch den Aufruf von *apt-get upgrade* installiert. Ob eine neue Version vorliegt, wird anhand der lokalen Paketlisten ermittelt. Deswegen ist es wichtig, immer ein *apt-get update* vor *apt-get upgrade* durchzuführen. Der Aufruf von *apt-get dist-upgrade* geht noch einen Schritt weiter. Mit diesem Kommando werden auch neue Pakete installiert und nicht mehr benötigte Pakete gelöscht. Zusätzlich gibt es bei *apt-get dist-upgrade* eine priorisierte Installationsreihenfolge, das heißt, wichtige Pakete werden zuerst aktualisiert.

Listing 14: Samba-Daemon läuft auf dem Testsystem

```
pi@raspberrypi:~ $ service smbd status
? smbd.service - LSB: start Samba SMB/CIFS daemon
(smbd)
  Loaded: loaded (/etc/init.d/smbd)
  Active: active (running) since Sun 2016-06-05
  12:17:11 UTC; 30min ago
  Process: 748 ExecStart=/etc/init.d/smbd start
  (code=exited, status=0/SUCCESS)
  CGroup: /system.slice/smbd.service
          +-850 /usr/sbin/smbd -D
          +-907 /usr/sbin/smbd -D
```

Listing 15: Minimale smb.conf

```
[global]
workgroup = WORKGROUP
security = user
encrypt passwords = yes

[Projekte]
path=/home/pi/projects
read only = no
```

passwd – a pi generiert werden. Jetzt müssen noch die Dienste *smbd* und *nmbd* neu gestartet werden:

```
service smbd restart
service nmbd restart
```

Nun kann man vom Hostrechner auf das freigegebene Verzeichnis zugreifen und Dateien ändern oder ablegen.

Server zum Mitnehmen

Als Entwickler kommt man nicht daran vorbei, seine Arbeit anderen Kollegen oder auch dem Kunden zu präsentieren. Da man aber nicht überall Zugriff auf einen Server hat und auch nicht alles lokal vorhalten möchte, bietet sich doch der Raspberry Pi als kleiner Entwicklungsserver an. Im Artikel wurde gezeigt, dass ein Zugriff ohne Tastatur und Monitor problemlos möglich ist. Auch eine Arbeit komplett ohne Netzwerk ist über ein USB-TTL-Kabel möglich – damit kann man auch in Umgebungen ohne Netzwerk effizient auf dem Raspberry Pi arbeiten. Auch die initiale Konfiguration ist komplett ohne Netzwerk möglich, hier muss lediglich in der *cmdline.txt* in der *boot*-Partition die serielle Schnittstelle aktiviert werden.

Wer auf dem Raspberry Pi auch unterwegs arbeiten möchte, der sollte ein USB-TTL-Kabel immer dabei haben. Auch ein leistungsfähiger Akkublock empfiehlt sich in diesem Fall. Denn nicht jeder Zug zum Beispiel bietet eine funktionierende Spannungsversorgung. Mit Rasbian Jessie steht ein vollständiges Linux-System zur Verfügung, damit sind dann schnell Projekte aus GitHub geladen oder in ein Git-Repository synchronisiert. Für die Entwicklung stehen sämtliche Programmiersprachen bereit. Wer ein eigenes Projekt mit dem Raspberry Pi umsetzen möchte, für den bietet sich Python an. Hierfür gibt es auch Bibliotheken für den GPIO-Zugriff. ■

Dr. Markus Stäuble
ist Informatiker, Conference Chair der Developer Week und Programmleiter Make beim Franzis Verlag. Neben Make beschäftigt er sich viel mit dem Thema Mobile und hat zu dessen Auswirkungen auf die Arbeitswelt promoviert.
https://github.com/mstaeuble

Links zum Thema

- CoolTerm
  http://freeware.the-meiers.org
- Eclipse
  www.eclipse.org
- Fritzing
  http://fritzing.org
- Geany
  www.geany.org
- GitHub
  https://github.com
- LibreOffice
  https://de.libreoffice.org
- LiClipse
  www.liclipse.com
- LXDE
  http://lxde.org
- Minecraft
  https://minecraft.net/de
- NOOBS Download
  http://downloads.raspberrypi.org/NOOBS_latest
- Paragon ExtFS
  https://www.paragon-software.com/de/home/extfs-windows
- PuTTY
  www.putty.org
- PyDev
  www.pydev.org
- Python
  https://www.python.org
- PyVmMonitor
  www.pyvmmonitor.com
- Raspbian, Installation des Images
  https://www.raspberrypi.org/documentation/installation/installing-images/README.md
- Raspberry Pi
  https://raspberrypi.org
- TightVNC
  www.tightvnc.com
- USB Image Tool
  www.alexpage.de/usb-image-tool
- Win32 Disk Imager
  https://sourceforge.net/projects/win32diskimager
- x11vnc
  www.karlrunge.com/x11vnc
- Xming X Server
  www.straightrunning.com/XmingNotes
- XQuartz
  https://www.xquartz.org
- Zugriff per USB-TTL-Kabel
  https://learn.adafruit.com/adafruits-raspberry-pi-lesson-5-using-a-console-cable

Foto: garagestock / shutterstock.com

OROCOMMERCE: E-COMMERCE IM B2B-BEREICH

Fokus B2B-Bereich

OroCommerce ist eine E-Commerce-Software speziell für den B2B-Bereich.

Gute und ausgereifte E-Commerce-Lösungen gibt es mittlerweile wie Sand am Meer. Trotz des enormen Überangebots gibt es mit OroCommerce (www.orocommerce.com) nun eine neue Lösung, die eine Daseinsberechtigung hat, weil sie sich ausschließlich auf den B2B-Bereich fokussiert und sich in Bezug auf die Zielgruppe gekonnt von den etablierten Systemen, die sich primär dem B2C widmen, abgrenzt.

Auch wenn aktuell im E-Commerce in der Regel vom B2C gesprochen wird, entdecken immer mehr kleine und mittelständische Unternehmen die Vorzüge des B2B-Geschäfts. Sie entwickeln E-Commerce-Strategien und setzen beim Vertrieb auf den digitalen Handel. Die große Herausforderung besteht darin, dass im B2B-Bereich weniger ein normaler Online-Shop benötigt wird. Es geht vielmehr darum, intelligente Beschaffungs-Tools zu entwickeln, die sich nahtlos in die IT-Infrastruktur der Unternehmen integrieren. Prozesse müssen automatisiert ablaufen. Durch diese Anforderungen ändern sich zwangsläufig auch die Anforderungen an die eingesetzte E-Commerce-Lösung. Ein schickes Design rückt plötzlich in den Hintergrund, wichtig ist aber beispielsweise die Verwaltung von kundenindividuellen Preisen und die Anbindung an Drittsysteme. Es bestehen gravierende Unterschiede in den Anforderungen zwischen B2C und B2B, die unter anderem technischer Natur sind.

Diesen Unterschieden zwischen B2C und B2B möchte sich OroCommerce annehmen und eine Lösung bereitstellen, die speziell auf den B2B-E-Commerce abzielt. Bei den Gründern und Machern von OroCommerce handelt es sich um diejenigen Personen, die auch schon Magento gegründet und erfolgreich gemacht haben.

Yoav Kuttner, ehemals CTO und Gründer von Magento, kümmert sich bei OroCommerce um die technischen Themen und die Entwicklung der Software. Zusätzlich ist seit einigen Monaten auch Roy Rubin, ebenfalls Gründer und ehemaliger Geschäftsführer von Magento, im OroCommerce-Team vertreten. Abgesehen von diesen beiden Persönlichkeiten haben weitere Personen das Lager gewechselt – weg von Magento und hin zu OroCommerce. Das Team rund um Yoav Kuttner weiß also, wie man eine E-Commerce-Lösung erfolgreich entwickelt und was bei der Entwicklung zu beachten ist.

Features im Überblick

Im Gegensatz zum klassischen B2C-E-Commerce, in dem immer ein spezieller Endkunde einkauft und sich seinen persönlichen Benutzeraccount einrichtet, agieren im B2B-E-Commerce Unternehmen. Diese Unternehmen bestehen wiederum aus vielen Mitarbeitern, die zwar einzelne Accounts benötigen, aber dennoch einer übergeordneten Firma zuge-

ordnet werden. OroCommerce hat ein entsprechendes Management von Corporate-Accounts inkludiert, wodurch das Anlegen von Unternehmen samt Einkäufern beziehungsweise Benutzern problemlos möglich ist.

Die Verwaltung von mehreren Shops beziehungsweise Websites auf Basis einer Instanz gehört wie bei Magento zu den Features von OroCommerce. Personalisierung ist im B2B-E-Commerce besonders wichtig, da den Kunden oftmals bestimmte Produkte oder ganze Produktsortimente zugewiesen werden. Das heißt, nicht jeder Kunde darf alle Artikel aus dem Produktsortiment kaufen beziehungsweise einsehen. OroCommerce erlaubt die Definition, welches Produktsortiment von welchen Kunden gekauft beziehungsweise angesehen werden darf.

Multiple Preislisten

Das direkte Hinterlegen eines Preises für einen Artikel ist im B2B-E-Commerce in der Praxis nicht möglich. Durch die Integration von multiplen Preislisten kann in OroCommerce ein ausgefeiltes Pricing abgebildet werden, denn im B2B-E-Commerce hat ein Artikel in der Regel sehr viele unterschiedliche Preise, die davon abhängen, welcher Kunde diesen Artikel beziehen möchte. Über die Eingabe einer Artikelnummer können viele Produkte direkt mit wenigen Klicks bestellt werden. Das ist ebenfalls ein sehr gern gesehenes B2B-Feature, da die Kunden oftmals wiederkehrend dieselben Artikel bestellen und somit anhand der Artikelnummer wesentlich schneller ans Ziel kommen.

Aufgrund komplexer Produkte ist ein direkter Kauf im B2B-Bereich oftmals unwahrscheinlich. Daher erlaubt OroCommerce einen Angebotsprozess, bei dem Produkte zuerst angefragt werden können. Auf Basis der Anfrage wird dann ein Angebot erstellt, das im nächsten Schritt in eine Bestellung konvertiert werden kann.

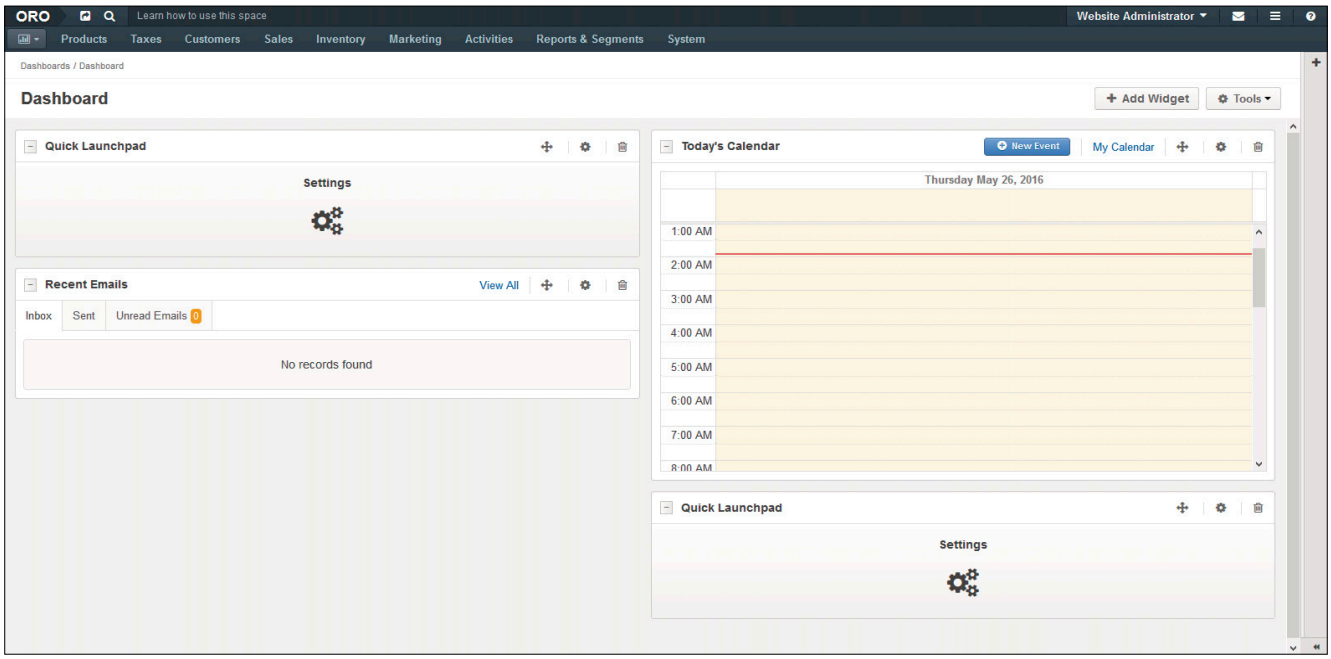
Im Frontend tritt OroCommerce in einem sehr schlichten und übersichtlichen Design an, das sich sehr gut als Basis für weitere Individualisierungen eignet. OroCommerce ist standardmäßig für das mobile Einkaufen optimiert. Zumindest in der Theorie, denn in der Realität gibt es in Sachen Responsive Webdesign noch das eine oder andere Problem. Dies kann jedoch an der frühen Version der Software liegen.

Systemanforderungen und Setup

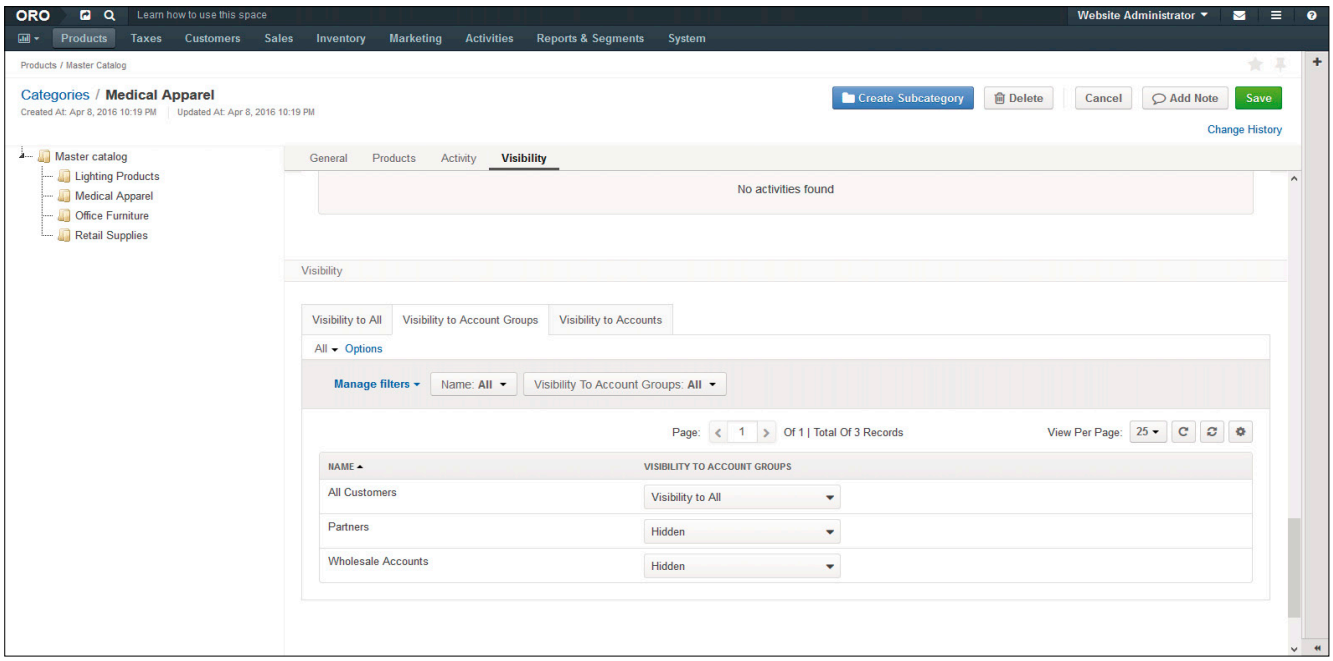
Neben einer kostenfreien Online-Demo, für die man sich unter https://www.orocommerce.com/demo registrieren kann, ist OroCommerce mittlerweile direkt via GitHub verfügbar. Unter https://github.com/orocommerce/orocommerce-application/releases findet man eine Auflistung der zur Verfügung stehenden Releases sowie Links zu den Installationsarchiven. Alternativ kann OroCommerce auch mittels Composer installiert werden. Hierfür muss man das Git Repository klonen und, sofern Composer bereits installiert ist, ein *composer install* ausführen. Weitere Informationen zur Installation erhält man bei Bedarf unter https://github.com/orocommerce/orocommerce-application.

Die Installation ist in wenigen Minuten abgeschlossen, da ein übersichtlicher Wizard Schritt für Schritt durch den Prozess führt.

Nach dem ersten Login wird man sofort viele Ähnlichkeiten mit Magento vorfinden. Man startet im Dashboard, das wichtige Informationen auf einen Blick zentral abbildet. Das Dashboard basiert auf sogenannten Widgets. Das sind kleine Helfer, die man sich auf das Dashboard legen kann. Zu den Widgets zählt beispielsweise *Recent E-Mails*, das die aktuellsten E-Mails aus dem Posteingang und Postausgang zeigt. Der Kalender *Today's Calendar* bildet den aktuellen Tag ab, und *Quick Launchpad* bietet eine Übersicht der wichtigsten Links. ▶



Die individuelle Konfiguration von Dashboards pro Nutzer ist problemlos möglich (Bild 1)

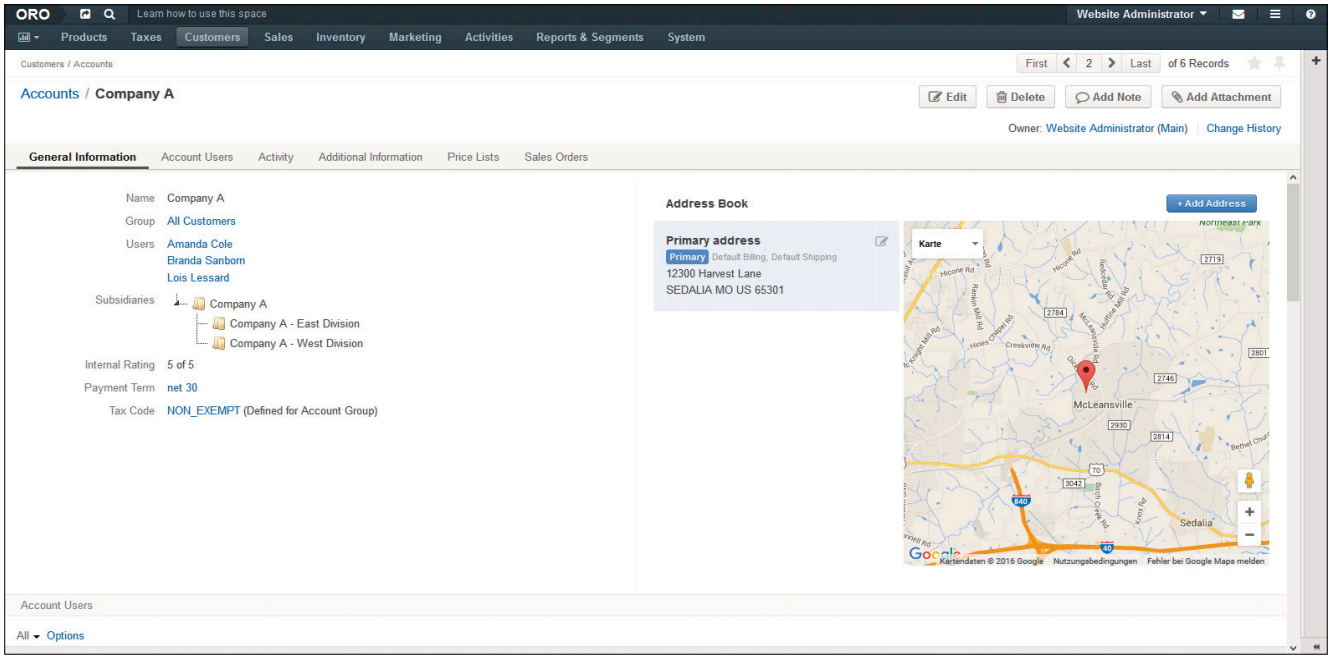


Zugriffsrechte pro Kategorie können auf Basis von Groups beziehungsweise Accounts definiert werden (Bild 2)

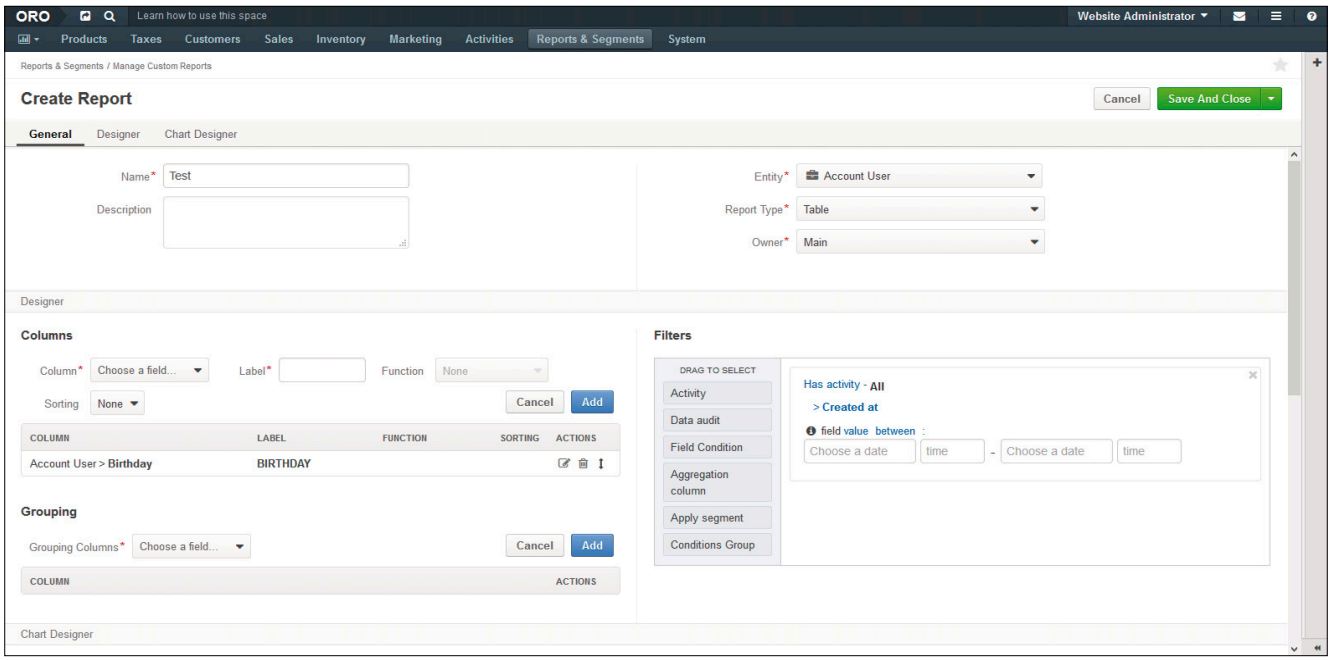
Ein sehr nützliches Feature ist die Option, beliebig viele Dashboards anzulegen (Bild 1). Denn jede Benutzerrolle hat tendenziell einen anderen Fokus und benötigt unterschiedlichste Kennzahlen und Widgets. Neben dem Dashboard existieren die übergeordneten Menüpunkte *Products, Taxes, Customers, Sales, Inventory, Marketing, Activities, Reports & Segments* und *System*. Dieser Aufbau erinnert sehr stark an Magento, und als Nutzer dieses Systems wird man sich in OroCommerce sehr schnell heimisch fühlen.

Das Management des Produktkatalogs teilt sich in die Bereiche *Master Catalog* und *Products*. Möchte man Kategorien anlegen, editieren oder löschen, ist man im *Master Catalog* richtig aufgehoben. Es handelt sich bei dieser Variante um die klassische Kategorieverwaltung. Jeder Kategorie können neben dem eigentlichen Namen auch weitere Informationen wie eine Kurz- beziehungsweise Langbeschreibung sowie ein kleines und ein großes Bild zugewiesen werden. Abgesehen davon lässt sich in dieser Maske definieren, welche Kundengruppen beziehungsweise Kunden überhaupt Zugriff auf diese Kategorie haben.

Innerhalb des Menüpunkts *Products* werden die einzelnen Produkte angelegt und verwaltet. Neben den gängigen Pro-



Die Kundenverwaltung ist funktionsreich und auf B2B-E-Commerce ausgelegt (Bild 3)



Reports können einfach und komplett individuell selbst erzeugt werden (Bild 4)

duktdaten ist ein besonderes Highlight die Unterstützung von Preislisten. Die Preisgestaltung kann bei OroCommerce im Vergleich zu beispielsweise Magento wesentlich feiner realisiert werden. Das ist speziell im B2B-E-Commerce wichtig und notwendig.

Auch bei den Steuern gibt es Ähnlichkeiten zu Magento. Wie auch in Magento ist die Steuerkonfiguration bei OroCommerce kompliziert und für den deutschen Markt unlogisch. Es besteht beispielsweise die Möglichkeit, den Steuersatz pro Stadt zu definieren. Was in den USA notwendig erscheint, sorgt in Deutschland eher für Verwirrung. Grundsätzlich können aber unter dem Aspekt der Internationalisierung Steuersätze und Steuerregeln für jedes Land individuell konfiguriert und hinterlegt werden.

Kundenverwaltung

Die Kundenverwaltung in OroCommerce erfüllt typische B2B-Anforderungen. Anstelle von Benutzeraccounts werden die Kunden in Accounts und Account Users aufgeteilt (Bild 2). Der Account symbolisiert eine Firma. Dieser Firma können beispielsweise Preislisten zugewiesen werden. An sich kann die Firma aber nicht einkaufen, daher besteht dieser Account aus Account Users, also den eigentlichen Benutzern, die sich im System anmelden. Durch dieses System kann innerhalb von OroCommerce problemlos eine Firma mit mehreren Usern abgebildet werden (Bild 3).

Abgesehen von diesem Feature sind auch die Möglichkeiten innerhalb der Kundenverwaltung sehr auf den B2B-Case ausgelegt. Aus dem OroCommerce-Backend können direkt E-Mails an Kunden versendet oder ein Event geplant werden. Wenn Sie also einen Kunden am Telefon haben, der vor dem Einkauf im Online-Shop noch schnell ein Angebot per Mail benötigt, können Sie dies direkt aus dem OroCommerce-Backend starten.

Angebote nehmen im B2B-E-Commerce einen hohen Stellenwert ein. Innerhalb des B2C können diese in der Regel vernachlässigt werden. Entweder der Kunde entscheidet sich für den Kauf eines Artikels, oder er bricht den Vorgang ab und verlässt den Online-Shop. Im B2B ist das jedoch aufgrund der komplexeren Produkte anders: Meist holt man sich zuerst ein Angebot ein und trifft die Kaufentscheidung später.

OroCommerce berücksichtigt dieses alternative Einkaufsverhalten und bietet *Shopping Lists* (Merkzettel), *Angebote* (Quotes) und *Bestellungen* (Orders) an. Innerhalb des Menüpunktes *Sales* können Sie diese Elemente verwalten. Zusätzlich gibt es noch einen Bereich namens *Requests for Quotes*, der sozusagen den Angeboten vorgelagert ist.

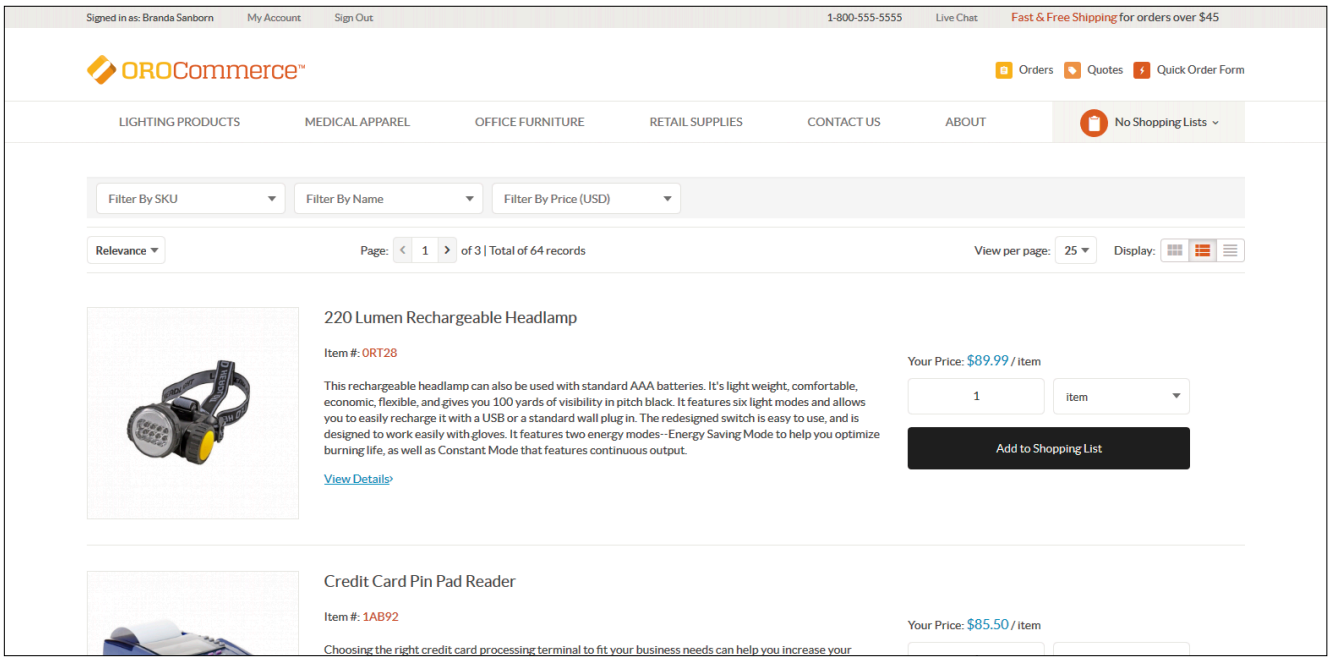
OroCommerce deckt daher sowohl im Frontend als auch im Backend den längeren und durch Anfragen beziehungsweise Angebote gekennzeichneten Einkaufsprozess ab und ermöglicht dem Online-Shop-Betreiber, diesen über umfangreiche Funktionen und Masken zu steuern.

Lagerverwaltung

OroCommerce bietet die Option, mehrere Lager zu verwalten. Es lassen sich relativ schnell neue Lager anlegen, was im Umkehrschluss bedeutet, dass der Lagerbestand eines Artikels pro Lager definiert werden kann. Auch dieses Feature erfüllt eine oftmals gewünschte Anforderung im B2B, und OroCommerce ist damit die einzige große E-Commerce-Lösung, die von Haus aus mehrere Lager unterstützt.

Die Marketing-Möglichkeiten sind innerhalb des Systems eingeschränkt. Unter anderem können Landingpages angelegt werden. Weitere Marketingmöglichkeiten und Features sucht man in OroCommerce aber vergeblich.

Der Menüpunkt *Activities* erlaubt die Verwaltung von Events. Denn Jeder Backend-User hat automatisch einen Kalender, den er mit Terminen befüllen kann. ▶



Ein schlichtes und übersichtliches Frontend sorgt für eine gute Customer Experience (Bild 5)

Das Reporting ist bei OroCommerce sehr umfangreich. Es werden jedoch keine vordefinierten Reports bereitgestellt, sondern Reports müssen zuerst definiert werden.

Dabei kann man auf sehr viele Informationen innerhalb des Systems zugreifen und über einen sogenannten Designer die Daten entsprechend filtern und aufbereiten. Dies bietet größtmögliche Flexibilität und ermöglicht die individuelle Konfiguration von Reports (Bild 4).

Wie in jedem Online-Shop darf eine umfangreiche Konfigurationsmöglichkeit des Systems nicht fehlen. Auch hier bietet OroCommerce viele nützliche Funktionen. Innerhalb dieses Punktes werden sich speziell die Magento-User wieder sehr wohlfühlen, denn die Konfigurationsmöglichkeiten und Prozesse erinnern sehr stark an Magento.

Das Frontend

Abgesehen von einem umfangreichen und gut strukturierten Backend kommt OroCommerce von Haus aus mit einem hübsch anzusehenden Frontend daher. Es ist sehr übersichtlich und schlicht gehalten. Im Vordergrund steht eine einfache Navigation und gute Benutzerführung, auf unnötige Eye-Candy-Elemente wurde bewusst verzichtet (Bild 5).

Darüber hinaus sind im Frontend wichtige Funktionen wie die Schnellbestellliste direkt integriert. Auch können Angebote und Bestellungen gleichermaßen verwaltet werden.

Eine sehr interessante Möglichkeit sind die Shopping Lists, also multiple Merkzettel. Sie bilden die Basis für die Erstellung eines Warenkorbs, der den Besucher anschließend in den Bezahlvorgang führt. Dieser wiederum ist relativ einfach strukturiert.

Ähnlich wie das Backend erinnert das Frontend, beispielsweise beim Benutzerkonto und beim Checkout-Prozess, stark an Magento. Das ist aber sicherlich nicht als Nachteil anzusehen, immerhin handelt es sich bei Magento um eine etablierte Lösung, die durchaus sehr gut durchdacht und gut strukturiert ist.

Wie auch bei klassischen B2C-Projekten wird auch in der Praxis das Frontend, ganz im Gegensatz zum Backend, je nach Projekt angepasst und individualisiert werden müssen. Die Basis, mit der OroCommerce ausgeliefert wird, ist aber durchaus brauchbar und eignet sich sehr gut als Basis für zukünftige Entwicklungen.

Fazit

Die E-Commerce-Software OroCommerce bezeichnet sich selbst als reine B2B-Lösung, und das wird definitiv beim Funktionsumfang, den Backend-Masken und der Ausrichtung der Software deutlich. Sofern man ein B2B-E-Commerce-Projekt realisieren möchte und dafür eine entsprechende Software benötigt, ist OroCommerce sicher einen Blick wert.

Bei dem im Artikel beschriebenen Programm handelt es sich um eine Beta-Version. Trotz des frühen Stadiums der Software macht diese jedoch bereits einen sehr guten Eindruck. Es bleibt abzuwarten wie sich OroCommerce zukünftig entwickelt. In Deutschland finden sich immer mehr Agenturen, die als offizieller OroCommerce-Partner auftreten, was als positives Zeichen zu werten ist. ■

Alexander Steireif
ist E-Commerce- und Magento-Experte. Vor seinem Wechsel zum Magento Gold Partner netz98 hat er lange Jahre die Magento-Agentur ITABS geführt. Er ist außerdem Autor von Fachbüchern und zahlreichen Artikeln zu E-Commerce-Themen.
info@alexander-steireif.com

DWX Developer Week

CODING

Die DWX bedankt sich bei:

- 1.600 Teilnehmern
- über 200 Experten
- über 30 Partnern

Aussteller & Sponsoren:

adesso business. people. technology. · adorsys · AIT · andrena OBJECTS Experts in agile software engineering · ASTRUM IT · brainLight LIFE IN BALANCE · byteAgenten · Configit · DevExpress · Fast Reports Reporting must be Fast! · GFU Cyrus AG · HANSER · HEINRICH & REUTER SOLUTIONS GMBH HeiReS · hemmersbach · JET BRAINS · MATHEMA · MAXIMAGO · Microsoft · n:aip Die care:manager http://www.naip.de · Saxonia Systems So geht Software. · SDX · SENACOR · SHS VIVEON · SOPHIST · SOS SOFTWARE SERVICE VALUE ADDED DISTRIBUTION · Telerik A PROGRESS COMPANY · TEXT CONTROL · ThingWorx A PTC Business · hello, technology · UID

Wir freuen uns auf ein Wiedersehen:
26.-29. Juni 2017!

developer-week.de

DeveloperWeek

ASP.NET, KNOCKOUT.JS UND SIGNALR

Effiziente Kommunikation

Single Page Applications stellen im Web zurzeit den aktuellen Technologie-Stack dar.

Webseiten, die ohne einen klassischen Seitenwechsel auskommen, werden als Single Page Application, kurz SPA, bezeichnet. Die Seite besteht also nur aus einem einzelnen HTML-Dokument, das die benötigten Ressourcen wie JavaScript und Stylesheet-Dateien sowie das komplette Markup beim ersten Aufruf lädt.

Das Ein- und Ausblenden von Seiteninhalten sowie das Laden und Speichern von Daten wird anschließend von JavaScript übernommen, sodass ein Neuladen der Seite nicht mehr notwendig ist.

Da die Inhalte jetzt dynamisch nachgeladen werden, entsteht eine Webanwendung in Form einer Rich-Client- beziehungsweise Fat-Client-Verteilung. Das heißt, eine SPA-Webanwendung ermöglicht die Reduzierung der Serverlast sowie die Umsetzung von selbstständigen Webclients, die auch eine entsprechende Offline-Unterstützung anbieten können. Weiterhin wird bei dieser Art von Webentwicklung die Client-Server-Kommunikation erheblich reduziert.

Es gibt inzwischen sehr viele Anwendungsfälle, in denen die Realisierung als Single Page Application für die Entwicklung sinnvoll ist. So ist zum Beispiel das Einsatzgebiet mit großen Benutzerzahlen typisch für SPA-Webanwendungen. Facebook, Google Mail, Google Maps und Twitter sind hier für uns Entwickler wahre Wegbereiter. Nützliche Umsetzungen sind auch im Bereich einer hohen Interaktivität gefragt, so wie sie bei Online-Spielen im Browser vorkommen.

SPA eignet sich auch besonders gut für die Entwicklung von Web-Apps, so kann die SPA-Webanwendung in die native mobile Anwendung eingebettet werden. Auch für kleinere Projekte, in denen sich die Menge der benötigten Logik, die für den Client, also in diesen Fall den Browser, umgesetzt wird, in Grenzen hält, eignet sich der Aufbau eine Single Page Application.

Da das Beispiel im Workshop mit einer Seite auskommt, benutzen wir hier die Single-Page-Application-Implementierung von ASP.NET. Weiterhin benötigen Sie zur Implementierung des SPA-Webclients noch ein entsprechendes JavaScript-Framework. Dann kann mit dem Smart Client im Browser gestartet werden. Für das Beispiel fällt die Wahl auf das Knockout.js-Framework.

Knockout

Bei Knockout.js handelt es sich um eine populäre freie und leichtgewichtige JavaScript-Bibliothek, die auf Datenbindung spezialisiert ist und das MVVM-Pattern implementiert. MVVM ist eine Variante des Model-View-Controller-Patterns und dient zur Trennung von Markup und Logik des User Interfaces.

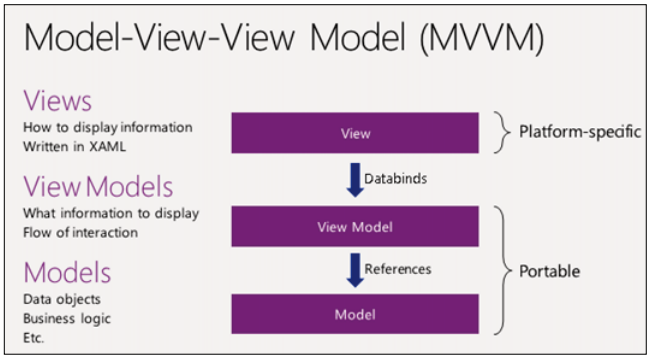
Warum ausgerechnet das Model-View-ViewModel-Pattern (MVVM)? Ganz einfach: In der Praxis wird sehr häufig versucht, weniger Abhängigkeiten zwischen Bedienoberfläche und Geschäftslogik zu erreichen. Somit erlaubt MVVM eine bessere Trennung von User-Interface-Design und User-Interface-Logik. MVVM ermöglicht hier durch die einfache Trennung von User Interface und Geschäftslogik eine entsprechende Auslagerung. Das MVVM-Pattern besitzt drei Hauptkomponenten: Model (Datenmodell), View (UI-Design) und ViewModel (UI-Logik).

Bild 1 zeigt die Abhängigkeiten des MVVM-Patterns. Die View nutzt das ViewModel, und die ViewModel-Schicht nutzt das Model. Umgekehrt bestehen keine Abhängigkeiten. Das heißt, die übergeordnete Komponente kennt nur die Untergeordnete. Somit kennt das Model weder das ViewModel noch die View. Die View ist somit für die Darstellung der Informationen zuständig.

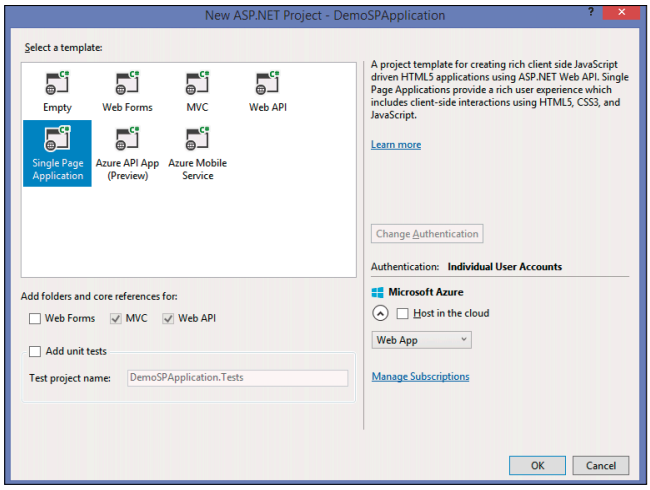
Jeder View wird eine ViewModel-Klasse zugeordnet, die die darzustellenden Daten und die Logik enthält. Somit werden praktisch alle Benutzereingaben mittels Binding an das ViewModel weitergeleitet und dort entsprechend weiterverarbeitet. Im Model werden die Klassen zusammengefasst, die sich um das Laden und Speichern der Daten kümmern. Es enthält somit auch die Geschäftslogik.

Somit ergibt sich über das MVVM-Pattern eine strikte Trennung von User Interface und User-Interface-Logik. Die grundsätzliche Struktur einer MVVM-Applikation, also auch einer Single Page Application mit Knockout.js, sollte immer folgende Ansätze besitzen:

- Pro View wird ein Interface angelegt, in dem die Funktion definiert wird, die das ViewModel benötigt, um die View steuern zu können.
- Das ViewModel erhält zur Laufzeit ein Objekt, das dieses Interface implementiert.



Das MVVM-Pattern: Schematische Darstellung (Bild 1)



Auswahl des Templates Single Page Application (Bild 2)

- Jede View erhält zur Laufzeit eine Instanz ihres ViewModels, um so auf die Daten zugreifen zu können.

Mit dieser grundsätzlichen Struktur lässt sich dann sehr einfach die Interaktion im MVVM umsetzen.

Der wichtigste Aspekt, der MVVM zu einem wirklich guten Muster macht, ist die Datenbindungsinfrastruktur. Durch das Binden von Eigenschaften einer View an ein ViewModel kommt es zur losen Kopplung zwischen den beiden, wodurch in einem ViewModel kein Code geschrieben werden muss, der eine View direkt aktualisiert.

Unter Knockout.js kann es sich beim Model entweder um ein einfaches JSON-Objekt oder um eine Konstruktorfunktion handeln. Um die Daten der Benutzereingabe an das Model zurückzuschreiben, stellt Knockout sogenannte Observables bereit. Sie sorgen auf jeden Fall dafür, dass jede Änderung am ViewModel oder am User Interface ohne zusätzlichen Methodenaufruf synchronisiert wird. Die View besteht bei Knockout aus klassischen HTML, sie beinhaltet jedoch noch zusätzlich ein *data-bind*-Attribut, das die Eigenschaften des Models mit den UI-Elementen in der Bedienoberfläche verknüpft.

SignalR

Bei SignalR handelt es sich um eine Client- und Server-Bibliothek, die Real-Time-Messaging zwischen Teilnehmern (Client und Server) ermöglicht und dabei verschiedene Protokolle mit einem Web-Socket-ähnlichen API unterstützt. Somit lässt sich SignalR für die Echtzeit-Kommunikation für browserbasierte Clients einsetzen. Die Kommunikation ist hierbei nicht nur auf Anforderung und Antwort beschränkt, sondern dauert so lange, bis sie ausdrücklich beendet wird. Die Kommunikation findet dauerhaft statt. Der browserbasierte Client kann mehrere Nachrichten an den Server senden, auf die dieser dann antwortet. Der Server kann wiederum asynchrone Nachrichten an den Client senden.

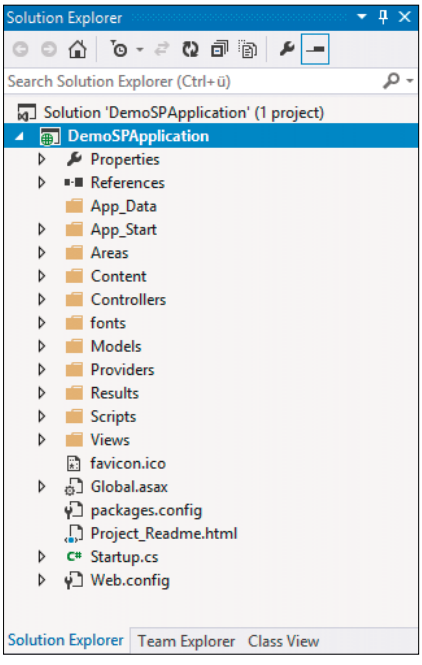
Zur Erstellung des in diesem Artikel beschriebenen Beispiels können Sie Visual Studio Community ab Version 2013 benutzen. Die Projektvorlage für ASP.NET finden Sie unter *Start, New Project …* oder über *File, New* und dann die Auswahl *Project*. Unterhalb des Visual-C#-Knotens stehen die Projektvorlage *Web* und die *ASP.NET Web Application* zur Auswahl.Wählen Sie die Projektvorlage aus und vergeben Sie als Vorbereitung für unsere Beispielanwendung den Namen *DemoSPApplication*. Sobald Sie das Dialogfenster *New Project* über *Ok* abgeschlossen haben, öffnet sich das Dialogfenster *ASP.NET-Projekttyp* zur näheren Spezifikation der zu erstellenden ASP.NET-Webapplikation.

Visual Studio bietet in diesem Bereich folgende Templates für die Erstellung einer Web- oder mobilen Applikation an:

- *Empty,*
- *Web Forms,*
- *MVC,*
- *Web API,*
- *Single Page Application,*
- *Azure API App (Preview),*
- *Azure Mobile Service.*

Wählen Sie hier für das Beispiel die Vorlage *Single Page Application* aus (Bild 2).

Visual Studio erstellt automatisch aus dem Template eine bereits lauffähige Webapplikation auf Basis von JavaScript und HTML. Diese Vorlage lässt sich auch für komplexere Client-Webapplikationen nutzen. Bild 3 zeigt die entsprechende Projektstruktur. Über die Taste [F5] oder das Startsymbol lässt sich die Single Page Application auch schon ausführen. Visual Studio startet hierfür im Hintergrund automatisch den Webserver IIS Express und präsentiert Ihnen die Startseite der Anwendung im Browser.



Projektstruktur der Anwendung (Bild 3)

In dem hier näher beschriebenen Beispiel soll eine einfache SPA-Webanwendung entstehen, die das Anzeigen von Daten aus einer Datenbank erlaubt. Die von Visual Studio erstellte Applikation wird hierbei als Infrastruktur verwendet und einfach um die benötigten Klassen erweitert. Um das Beispiel möglichst kompakt und überschaubar zuhalten, wird eine Funktion realisiert, um die Daten auf der Webseite aufzurufen. Die benötigten Daten werden für die Anzeige per Datenbankzugriff über SignalR zur Verfügung gestellt.

Aus Sicht des MVVM-Entwurfsmusters benötigen Sie für das Beispiel also ein Model, das als Datencontainer ▶

Listing 1: SignalR in der Startup-Datei

```
[assembly: OwinStartup(typeof(DemoSPApplication.
Startup))]
namespace DemoSPApplication {
  public partial class Startup {
    public void Configuration(IAppBuilder app) {
      app.MapSignalR();
    }
  }
}
```

fungiert, die View zur Visualisierung der Daten und das ViewModel, um Model und View miteinander zu verknüpfen. Um das MVVM-Entwurfsmuster und SignalR benutzen zu können, müssen Sie diese Dateien erst einmal im Projekt zur Verfügung stellen.

Um SignalR und Knockout.js im Projekt verwenden zu können, benötigen Sie die entsprechenden Dateien als Referenz. Am einfachsten geht das Einbinden über NuGet in Visual Studio. Wählen Sie hierfür im Kontextmenü des Solution-Explorers den Punkt *Manage NuGet Packages for Solution …* aus. Tragen Sie in das Suchfeld *knockout* beziehungsweise *SignalR* ein und fügen Sie Ihrem Projekt die benötigten Referenzen hinzu (Bild 4).

Nach der Installation müssen Sie die Klasse *Startup.cs* in der Solution mit dem Code aus Listing 1 überschreiben. Damit weisen Sie ASP.NET an, SignalR in der Anforderungspipeline zu verwenden. Diese Klasse stellt jetzt ein Open-Web-Interface für .NET-Startup (OWIN) bereit und ermöglicht die bidirektionale Kommunikation.

Weiterhin kann man bei SignalR auswählen, ob das Verbindungs-Pattern Persistent Connection oder Hub verwendet werden soll. Da das Hub-Interface die Ereignisse *OnConnected, OnDisconnected* und *OnReconnected* anbietet, sollte die Kommunikation über SignalR-Hub implementiert werden.

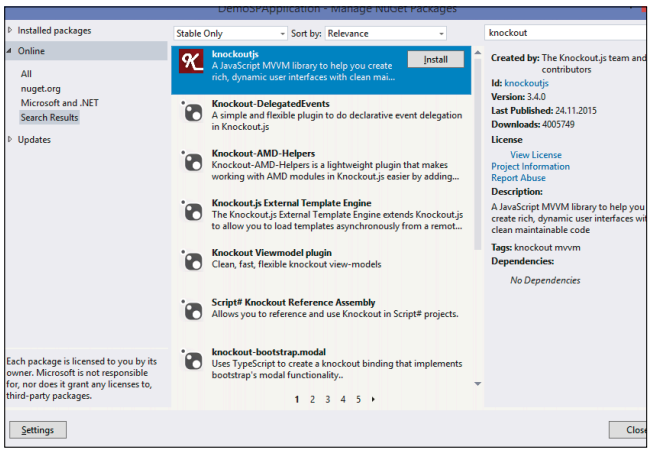
Der SignalR-Hub erhält die eingehende Anforderung und leitet die Nachricht an den oder die Clients weiter. Hierfür ruft der Client eine öffentliche Methode auf dem Hub auf. Daher kann im einfachsten Fall der Code für den Hub wie ein Webdienstaufruf aussehen.

Der benötigte Hub stellt das Mittelstück unserer SPA-Applikation da. Der Hub ist verantwortlich für den Empfang von Eingabe und Ausgabe. Im Beispiel soll auf einfache Art der Zugriff auf die Beispieldatenbank von Microsoft AdventureWorksLT2008 ermöglicht werden. Da die öffentliche Methode in der Hub-Klasse auf die Tabelle *SalesOrderHeader* zugreifen soll, wird im Beispiel die Hub-Klasse wie der Oberbegriff *SalesOrder* bezeichnet. Um die Hub-Klasse zu erstellen, wählen Sie im Kontextmenü der Solution *Add …, New Item …* aus. Klicken Sie im Knotenpunkt *Web* auf *SignalR* und wählen Sie die Vorlage *SignalR Hub Class (v2)* aus und vergeben Sie den Namen *SalesOrder* (Bild 5).

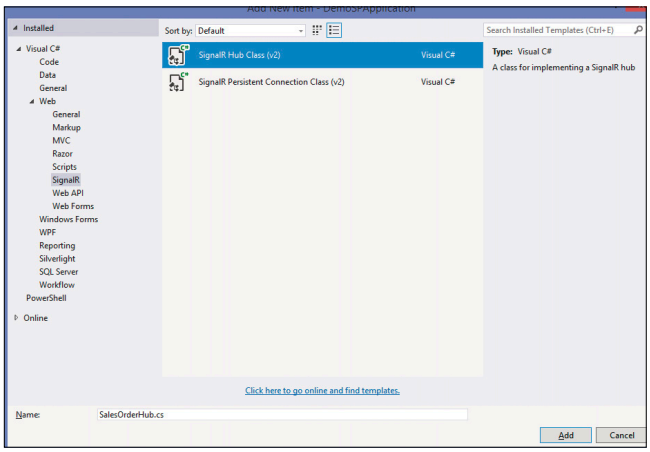
Listing 2 zeigt den benötigten Code für das Öffnen der Datenbank und das Abfragen auf die Tabelle. Die Methode *GetSalesOrderHeader* wird hier von den Clients aufgerufen. Sie öffnet den Zugriff auf die Datenbank und holt über das *Select*-Statement einen Datensatz mit der Angabe der *SalesOrderID* aus der Tabelle mit der *SalesOrderNumber* SO71815. Über die Methode *Clients.All.updateSalesOrder( )* werden alle verbundenen Clients aktualisiert.

Der Zugriff auf die Datenbank in der Hub-Klasse erfolgt ganz einfach über das Paket WebMatrix.Data. Dieses Paket enthält bereits Klassen, die Sie beim Öffnen und Abfragen an eine Datenbank unterstützen. Der angegebene Name ist der zu öffnenden Datenbank zugeordnet und kann eine SDF- oder MDF-Datenbankdatei sein, die im *App_Data*-Ordner der Solution abgelegt ist. Um Knockout.js zu verwenden, müssen Sie die JavaScript-Bibliothek in Ihre Webseite einbinden. Weiterhin stellt Knockout seine Funktionen über ein globales JavaScript Objekt *ko* zur Verfügung.

Somit können Sie Elemente auf einer Webseite entweder über einen direkten Zugriff auf das DOM oder mit Hilfe einer weiteren Bibliothek wie etwa jQuery bearbeiten. Das Document Object Model (DOM) beschreibt HTML- und XML-Dokumente. Für JavaScript stellt das DOM den Standard für den Zugriff auf Tags, Attribute und Inhalte von HTML-Seiten und XML-Dokumenten dar. Hierbei werden alle Elemente zu Ob-



Referenz über NuGet hinzufügen (Bild 4)



Auswahl der SignalR Hub Class (Bild 5)

Listing 3: Datenbindung mit Knockout

```
@{
 Page.Title = "Knockout";
}
@section script{
<script type="text/javascript">
 var viewModel;
 var salesOrderHub;
 $(function() {
  viewModel = {
   salesOrderId: ko.observable() };
  viewModel.canGet = function() {
   salesOrderHub.server.GetSalesOrderHeader(); };
  ko.applyBindings(viewModel);
  productsHub = $.connection.productHub;
  productsHub.client.updateSalesOrder =
  function (canGet) {
   viewModel.salesOrderId;
   $('span[data-bind="text: salesOrderID"]').
    effect('highlight', { color: '#9ec6e2' },
    3500); };
  $.connection.hub.start();
 });
</script> }
<div id="products">
 <h3></h3>
 <select id="categories" data-bind="options: viewModel.
 salesOrderID, optionsValue: 'SalesOrderID'"></select>
 <table>
  <tbody data-bind="foreach: viewModel.salesOrderID">
   <tr>
    <td><span data-bind="text:salesOrderID">
    </span></td>
   </tr>
  </tbody>
 </table>
</div>
```

jekten, die dynamisch aufgerufen, verändert, hinzugefügt und gelöscht werden können. Der große Vorteil von Knockout und MVVM ist, dass Sie selbst keinen Code zum Bearbeiten von DOM-Elementen oder Benutzeraktionen zu implementieren brauchen. Interagieren Sie einfach mit dem ViewModel-Objekt. Um alles Weitere kümmert sich Knockout.

Knockout stellt hierfür sogenannte Observables bereit, die dafür sorgen, dass jede Änderung am ViewModel oder am UI ohne einen zusätzlichen Methodenaufruf synchronisiert wird.

Die Datenbindung mit dem ViewModel lässt sich in Knockout einfach mit dem bereitgestellten HTML-Attribut *data-bind* konfigurieren und mit dem Aufruf der Methode *applyBindings* realisieren. Auch die Sichtbarkeit von CSS-Klassen oder CSS-Stilen kann durch Datenbindung beeinflusst werden. Hierfür stehen Eigenschaften wie *visible, css* und *style* zur Verfügung. Des Weiteren ermöglicht das *observableArray*-Objekt die Anzeige dynamischer Listen.

Im Beispiel wird das MVVM-Pattern gemeinsam in einer HTML-Seite verpackt. Das Model stellt das JSON-Objekt dar, die passende View wird durch die *data-bind*-Attribute in HTML angegeben und das Binden der Daten erfolgt über *Ko.applyBindings(viewModel)*. Listing 3 zeigt den übersichtlichen Aufbau von Model,ViewModel und Binding.

Es gibt im Beispiel eine einfache Schaltfläche, die das Ergebnis der Datenbankabfrage über den SignalR-Hub anfordert. Wie Sie an der Struktur erkennen können, ergänzen sich Knockout und SignalR sehr gut. Es dürfte Ihnen nicht schwer fallen, die Datenabfrage zu erweitern, anzupassen und darzustellen. Weiterhin können Sie auch alle Datenbankoperationen wie *Create, Read, Update* und *Delete* (CRUD) ohne größeren Aufwand umsetzen.

Die erzeugte Knockout.cshtml-Seite wird im Projekt unter den Ordner *Views/Home* abgelegt und im *HomeController* über die *ActionResult*-Methode aufgerufen. Mit Knockout steht Ihnen ein einfaches Framework für Single Page Applications zur Verfügung. Das Hauptaugenmerk des Frameworks liegt in der Nutzung von MVVM und Datenbindung. ■

Listing 2: Die Hub-Klasse SalesOrder.cs

```
using Microsoft.AspNet.SignalR;
using WebMatrix.Data;
namespace DemoSPApplication {
  public class SalesOrderHub : Hub {
    public void GetSalesOrderHeader() {
      using (var db = Database.Open
      ("AdventureWorksLT2008")) {
        var dbItem = db.QueryValue("Select
        SalesOrderID FROM SalesOrderHeader WHERE
        SalesOrderNumber ='SO71815'");
        Clients.All.updateSalesOrder();
      }
    }
  }
}
```

Daniel Basler
ist Senior Consultant für Microsoft-Technologien und beschäftigt sich darüber hinaus mit Datenbanken und Compiler-Bau. Er ist Autor zahlreicher Artikel über diese Themenkomplexe.

GRAFIK FÜR ENTWICKLER: COMICS

Komische Bilder

Comics gibt es nicht erst seit gestern. Und heute sind sie in allen Medien angelangt.

Ein Comic ist im weitesten Sinn eine gezeichnete Geschichte, die mit Schrift kombiniert ist. Dabei können die Texte in erklärender Form unter dem Bild stehen, oder sie befinden sich als verbale oder gedankliche Äußerungen in Sprech- und Denkblasen. Auch Geräusche oder Lautäußerungen der Protagonisten werden sprachlich aufgegriffen und in Textform dargestellt (Onomatopoesie). Die einzelnen Bilder, die die Geschichte erzählen, sind in der Regel auf Panels angeordnet, sie zeigen also eine Umrandung.

Zu den bekanntesten Comics gehören die Geschichten von Walt Disney und dem Zeichner Ub Iwerks, die bereits Ende der 1920er Jahre die Kult-Comicfigur Micky Maus schufen. Noch heute ist die Figur brandaktuell und wohl in jedem Kinderzimmer dürfte das eine oder andere Heft über Micky Maus, Donald Duck und anderen Comicfiguren der beiden Künstler zu finden sein – mittlerweile tummeln sich Walt Disneys Kultfiguren nicht nur auf dem Papier oder in Filmen, sondern auch im Internet (Bild 1). Soll also eine Anwendung im Comic-Stil entstehen, lohnt sich ein Blick zurück.

Vorläufer des Comics

Erzählende Bildfolgen gab es bereits in der Antike, und diese können als Vorläufer der heutigen Comics betrachtet werden. So zeigt beispielsweise das Grab des Menna, einem Ackerschreiber des Königs aus der 18. Dynastie, Alltagsszenen wie etwa das Dreschen von Getreide (Bild 2).

flickr, Horus3, Tomb of Menna/Menena TT 69

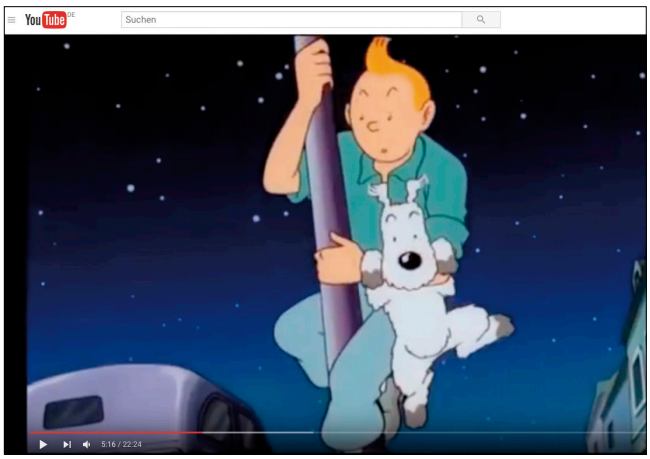
Grab des Menna: Erste Bilderfolgen entstanden bereits in der Antike (Bild 2)

Weitere sequenzielle Bildfolgen sind auf der Trajansäule zu sehen. Ein spiralförmiger Streifen, der sich 23-mal um die Säule wickelt, zeigt frühe Kriegsszenen in einer außerordentlichen Präzision: 2500 Menschen sind auf dem Fries zu entdecken, deren Darstellung aufschlussreiche Informationen über die Bekleidung und über verschiedene Kriegsformen und -mittel im zweiten Jahrhundert liefert (Bild 3). Ein weiteres Beispiel alter Bilderfolgen zeigt der Teppich von Bayeux

Micky-Maus.de: Die klassische Darstellung eines Comic-Strips ist beispielsweise in den Micky-Maus-Heften zu finden (Bild 1)

Erste Bildergeschichten: Die Trajansäule wurde in den Jahren 112/113 n. Chr. zu Ehren des Kaisers Trajan in Rom erbaut (Bild 3)

Eine Zeichnung aus dem Satire-Magazin Punch (1909) (Bild 4)



Tim und Struppi sind heute auch in YouTube zu finden (Bild 5)

aus Frankreich, auf dem die Eroberung Englands aus dem Jahr 1066 zu sehen ist. Auch in Japan bedienten sich bereits im 12. Jahrhundert Mönche dieser Vermittlungstechnik und zeichneten auf Papierrollen Bildergeschichten.

Durch die Erfindung des Buchdrucks wurden Bildergeschichten einem breiteren Publikum zugänglich gemacht. Die Themen waren hier oftmals sozialkritischer Natur. Im 18. Jahrhundert wurden dann lustige Zeichnungen erstmals als Comic Print bezeichnet. Ende des Jahrhunderts festigte sich der Begriff Comic und fand in unterschiedlichen Formen weitere Verbreitung, beispielsweise durch das englische Satire-Magazin Punch (Bild 4).

Beginn des modernen Comics

Als Vater der heutigen Comics gilt der Schweizer Zeichner Rodolphe Töpffer (1799–1846), der als Erster seine Bilder in Panels setzte. Diese Technik war dann auch auf den Münchener Bilderbögen zu finden. Diese sind in der zweiten Hälfte des 19. Jahrhunderts erschienen und wurden von verschiedenen, teils heute noch bekannten Künstlern gestaltet, unter anderem von Wilhelm Busch (1832–1908). Dessen Bildergeschichten zu Max und Moriz stehen heute nicht nur in Buchform, sondern auch auch als App zur Verfügung.

Ende des 19. Jahrhunderts erschienen in den USA Comics in Zeitungen. Hier erfreute sich The Yellow Kid, eine Comicfigur von Richard Felton Outcault (1863–1928), großer Beliebtheit. Im Gegensatz zu den einzelnen Bildern von The Yellow Kid lieferten die Comics The Katzenjammer Kids von Rudolph Dirks (1877–1968) Bildergeschichten, die eine Handlung und wichtige Comic-Elemente wie Panels und Sprechblasen zeigten. Zu Beginn des 20. Jahrhunderts entwickelten sich nebeneinander verschiedene Comic-Stilrichtungen in einzelnen Ländern, die auch unterschiedliche Themen aufnahmen.

In Europa begründete beispielsweise der belgische Zeichner Hergé (Georges Prosper Remi, 1907–1983) mit Tim und Struppi im Jahr 1929 den neuen Comicstil Ligne claire. Anders als ältere Comic-Zeichnungen liefert diese Richtung vereinfachte Formen, die möglichst wenig Textur wie Schattierungen oder Schraffuren zeigen, und ist in klaren Farben gehalten. Heute sind Tim und Struppi in fast allen Medien vertreten (Bild 5).

In Japan übernahmen die Zeichner Sprechblasen auch in ihre Comics. Bekannte Mangakas, Zeichner und Schreiber von Mangas, waren Okamoto Ippei (1886–1948) und Rakuten Kitazawa (1876–955). Mangas haben eine lange Geschichte und Tradition, Vorläufer gab es bereits im 8. Jahrhundert. Nicht jeder Manga zeigt das uns heute bekannte Kindchenschema; dies kam erst in etwa der zweiten Hälfte des letzten Jahrhunderts in Mode. Heute gibt es verschiedene Manga-Stile, die sich an ihrer Zielgruppe orientieren: Diese reichen vom Kleinkind-Manga bis hin zum Silver Manga für Senioren (Bild 6).

In der ersten Hälfte des 20. Jahrhunderts tauchten die ersten Superhelden in den gezeichneten Abenteuern auf und machten damit die unterschiedlichen Comic-Stile einer breiteren Bevölkerungsschicht zugänglich. Hier waren die beiden US-Amerikaner Joe Shuster (1914–1992) und Jerry Siegel (1914–1996) mit ihrer Figur Superman federführend: Sie schufen kurzerhand ein eigenes Magazin, genannt Sci- ▶

Frühes Manga: Eine Comiczeichnung des Japanischen Künstlers Rakuten Kitazawa aus dem Jahr 1902 (Bild 6)

ence Fiction, und teilten sich die Arbeit: Shuster zeichnete und Siegel schrieb die Geschichten. So entstand zunächst ein böser Glatzkopf, der mit seinen übersinnlichen Fähigkeiten die Weltherrschaft zu erlangen versuchte. Später wandelte sich der Comic-Held zum Guten; mit dabei war nun ein weiterer Zeichner, Tony Strobl (1915–1991). Im Frühjahr 1938 wurde im National Publications Verlag das erste Action Comic verlegt – mit großem Erfolg. Der erste Superheld war geboren (Bild 7).

Weitere Superhelden folgten: Ob Batman, Spider-Man, Iron Man oder Catwoman; sie alle bringen magische Kräfte mit, kämpfen gegen Bösewichte und zeigen ein äußerst imposantes Äußeres mit hohem Wiedererkennungswert.

Comic, Cartoon oder Karikatur?

Der Comicstil gilt seit Beginn der 1980er Jahre als eigene Kunst- und Kommunikationsform. Daneben gibt es noch Cartoons, Karikaturen oder andere Darstellungsformen, die ebenfalls gezeichnete Bilder zeigen. Hier ist eine scharfe Abgrenzung kaum möglich. Beispiel: In der Frankfurter Allgemeine erscheint regelmäßig ein Comic des Künstlers Flix. Um es zu finden, muss man dem Pfad *Home, Feuilleton, Cartoons* und *Comic Glückskind* folgen (Bild 8). Ist also ein Comic eine Unterform des Cartoons?

Comic, Comic-Strip oder Bildgeschichte: Generell handelt es sich bei einem Comic um eine Abfolge von meist kolorierten Zeichnungen. Das Comic erzählt eine Geschichte, die über eine längere Bildstrecke gehen kann und meist Themen aus dem Alltag aufgreift.

Aus dem Comic heraus hat sich die Graphic Novel entwickelt: Publik machte diesen Begriff der US-amerikanische Comic-Zeichner Will Eisner: So betitelte er sein Buch »Ein Vertrag mit Gott« (1978) bereits auf dem Cover als Graphic Novel und etablierte diese neue Kategorie der umfangreicheren und thematisch anspruchsvolleren Bildergeschichten mit Text auf dem Markt. Graphic Novels greifen oft Themen aus der Literatur auf und werden in Buchform angeboten (Bild 9).

Ein Cartoon ist meist in Magazinen oder Zeitungen zu finden. Dabei handelt es sich ebenfalls um eine komische, witzige oder auch groteske Zeichnung, die, wie ein Comic, meist eine alltägliche Situation darstellt. In Deutschland sind politische Aussagen im Cartoon seltener zu finden. Eine solche Witzzeichnung hat eine Pointe und funktioniert meist ohne Worte.

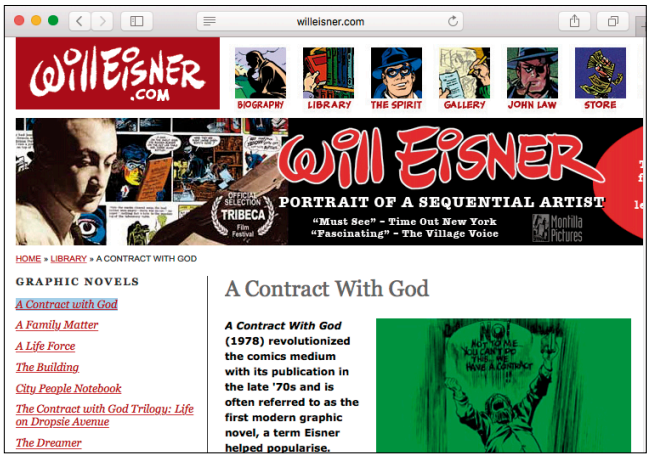
Comic oder Cartoon? Die einzelnen Formen der lustigen Zeichnungen sind schwer voneinander abzugrenzen (Bild 8)

Nicht narrativ, also ohne Geschichte, funktioniert hingegen die Karikatur. Sie stellt Personen oder gesellschaftliche Anliegen kritisch überzeichnet dar und liefert somit eine klare Aussage. Karikaturen sind in fast allen Medien angesiedelt und auch in Satire-Magazinen zu finden, etwa der französischen Satirezeitschrift Charlie Hebdo, die seit 1970 regelmäßig erscheint. Eine Karikatur Mohammeds bot schließlich im Jahr 2011 Anlass zu einem Brandanschlag (Bild 10).

Eigene Comics per App

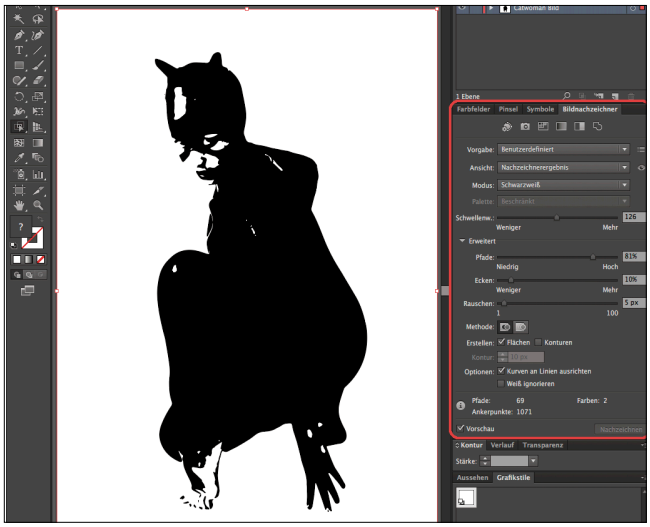
Doch nicht nur auf Papier, im Film oder im Netz hat sich diese Stilrichtung etabliert. In der Fülle der unterschiedlichen Comic-Apps gilt es grundsätzlich, drei Kategorien voneinander zu unterscheiden: Apps, deren Grafik im Comicstil gehalten ist, jene, die fertige Bildergeschichten zeigen, und solche, die beim Erstellen von eigenen Comics helfen. Wer also eige-

Wer wird siegen? Die App von Warner Bros. International Enterprises haucht Superman neues Leben ein (Bild 7)



Ein Vertrag mit Gott war eine der ersten Graphic Novels von Will Eisner (Bild 9)

Die Website des französischen Satiremagazins Charlie Hebdo, das Opfer eines Terroranschlags wurde (Bild 10)



Handzeichnungen oder Fotos lassen sich in Illustrator über den Bildnachzeichner vektorisieren (Bild 12)

ne, kleine Bildergeschichten für den persönlichen Gebrauch kreieren möchte, kann das schnell und bequem mit den beiden folgenden vorgestellten Apps meistern:

Die App Comic Life von plasq LLC liefert ihren Anwendern das nötige Werkzeug, um aus eigenen Fotoserien Comic-Geschichten zu basteln. Für die selbstgebackene Geschichte und die Äußerungen der Protagonisten stehen verschiedene Sprechblasen und Textfelder bereit. Einzelne Elemente können auf der Fläche recht flexibel angeordnet und skaliert werden. Zudem lassen sich Rahmen und Text mit entsprechend genrekonformen Effekten versehen.

Ergänzend liefert derselbe Hersteller die App Comic Touch 2. Hier gibt es neben den obligatorischen Sprechblasen und Schriftstilen unterschiedliche Filter zum Vereinfachen der Formen, die den Bildern einen mehr oder weniger comicartigen Stil verpassen (Bild 11).

Um wirklich professionelle Comics zu erstellen, bedarf es verschiedener Fähigkeiten; mit wenigen Mausklicks kommt hier auch der ambitionierte Illustrator nicht ans Ziel. So gilt es zunächst eine schlüssige Geschichte mit Pointe zu entwickeln, die von einer oder mehreren Figuren getragen wird. Diese sollten jedoch deutliche Charakterzüge zeigen – bezüglich der Handlung wie auch ihres Äußeren.

Bei einer Comic-Figur greift der Künstler bestimmte, der Zielgruppe bekannte Eigenschaften auf und stellt sie bildnerisch dar. Dabei werden Details bewusst überzeichnet, etwa der besonders dicke Bauch, die riesigen Augen nach dem Kindchenschema für das Manga oder die allzu lange Nase. Erst dadurch entsteht die gewisse Komik, natürlich unterstützt durch die richtige Geschichte.

Durch einfaches Vektorisieren von Personen auf Fotos ist dies natürlich nicht zu erreichen. Hier fehlt das gewisse Etwas, eben der Charakter. Comic-Zeichner arbeiten in der Regel zuerst ganz klassisch mit Stift und Papier. Erst dann werden die Zeichnungen digitalisiert, verändert, koloriert und finalisiert.

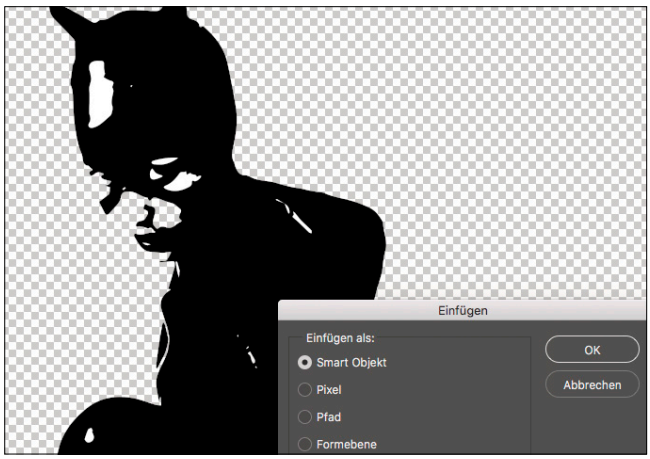
Die App Comic Touch 2 baut aus Foto-Inhalten comicartige Formen (Bild 11)

Umsetzung

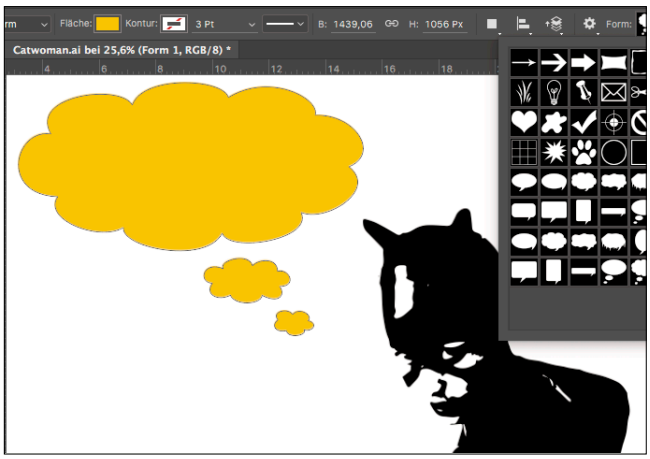
Zum Erstellen von Grafiken im Comic-Stil bieten sich professionelle Programme an, allem voran die beiden Lösungen aus der Adobe Cloud CC, Illustrator oder Photoshop.

In Illustrator können Handzeichnungen oder auch Fotos per Bildnachzeichner in Vektorgrafiken werden. Soll die Grafik nachträglich koloriert werden, eignet sich am besten der Modus Schwarz-Weiß. Der Schwellenwert regelt dabei die Größe der schwarzen Bereiche (Bild 12).

Nach dem Umwandeln der vektorisierten Vorlage lassen sich die einzelnen Formen dann wortwörtlich Punkt für Punkt bearbeiten. So entstehen beispielsweise glatte Umrisse, wenn gewünscht, oder der Künstler kann gewisse Details weiter ausarbeiten. Mit den Interaktiv-Malen-Werkzeugen ist es dann möglich, Bestandteile der Zeichnung einzufärben. Verschiedene Hilfen, etwa die Farbhilfe oder die Color-Schemata, liefern Funktionen zur Wahl und Zusammenstellung der Farben. Sprechblasen und Text können dann in Photoshop auf einer neuen Ebene über die Grafik gelegt werden. Für ▶



Per Copy and Paste erscheint die Grafik in Photoshop als bearbeitbares Smartobjekt (Bild 13)



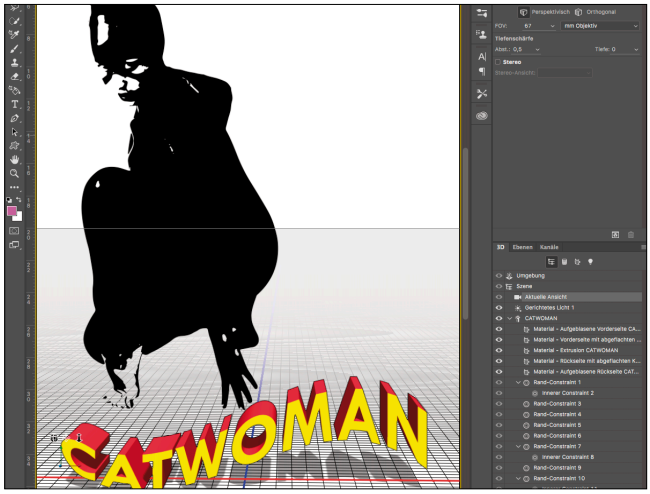
Sprechblasen entstehen in Photoshop über das Eigene-Form-Werkzeug (Bild 15)

den Import der Grafik bieten sich mehrere Möglichkeiten an: Wird eine Illustrator-Datei in Photoshop geöffnet, öffnet sich zunächst der Dialog *PDF importieren*. Hier stehen die Einstellungen zur Bildgröße und zur Auflösung bereit.

Besser geeignet sind die beiden Befehle *Platzieren und einbetten* oder *Platzieren und verknüpfen*. Bei der ersten Variante wird die Zeichnung zum Bestandteil der Photoshop-Datei. Bei einer verknüpften Grafik hingegen verweist Photoshop auf die externe Illustrator-Datei.

Zudem kann eine Grafik einfach über die Zwischenablage von Illustrator aus in ein Photoshop-Dokument kopiert werden. Als Smart Object eingefügt, kann es nachträglich verlustfrei skaliert und in Illustrator weiterbearbeitet werden (Bild 13). In Photoshop angelegter Text kann über die Optionsleiste bequem verformt werden.

Zusätzlich bietet es sich an, den Text dreidimensional zu gestalten. Dazu stellt Photoshop den Arbeitsbereich 3D bereit, der dem Anwender mittlerweile umfangreiche Werkzeuge und Optionen liefert (Bild 14). Über einen Ebenenstil erscheint zusätzlich eine Kontur um die Buchstaben.



Text kann im 3D-Arbeitsbereich von Photoshop gestaltet werden (Bild 14)

Sprechblasen gelingen mit dem Eigene-Form-Werkzeug. Eine Auswahl an entsprechenden Formen liefert die Bibliothek Sprechblasen (Bild 15).

Doch auch verschiedene Open-Source-Lösungen liefern flexible Werkzeuge und Funktionen, die durchaus professionellen Ansprüchen genügen. So eignet sich etwa Inkscape zum Erstellen von Vektorgrafiken.

Fazit

Unter den verschiedenen Kunstrichtungen ist gerade der Comicstil besonders gut zur Gestaltung verschiedener digitaler Anwendungen geeignet. Im Vergleich zu anderen Kunstrichtungen zeigen sich Comics in ihrer Machart besonders plakativ. Sie erzählen eine Geschichte, die dem Betrachter direkt zugänglich ist und zudem mit Text angereichert werden kann; speziell auch über die Zuordnung von Sprech- und Gedankenblasen zu einzelnen Charakteren.

Bedient man sich dieses Stils, ist ein Blick auf die Anfänge der Comics sinnvoll. Hier zeigt sich, wie bestimmte Richtungen entstehen und welche Charaktere sich bis heute halten.

In den letzten Jahrzehnten haben sich viele verschiedene Comicstile etabliert, die dieser Artikel nicht anspricht – nicht ansprechen kann, da die Fülle der Möglichkeiten auf nur wenigen Seiten kaum Platz findet. Die kleine vorgestellte Auswahl soll jedoch zeigen, wie komplex das Thema Comic ist. Sicher liefert ein Streifzug durch einschlägige Buchläden oder das Internet weitere Anregungen, die eventuell bei dem ein oder anderen eigenen Projekt helfen. ■

Katharina Sckommodau
arbeitet als freiberufliche Autorin, Grafikerin und Dozentin, unter anderem für die Akademie der Bayerischen Presse und für Macromedia. Sie veröffentlicht regelmäßig Beiträge in renommierten Fachzeitschriften.

USER-EXPERIENCE-DESIGN UND SCRUM

Ein gutes Doppel

Wie sich UX-Design erfolgreich in den Scrum-Prozess integrieren lässt.

Scrum als agile Projektmanagement-Methode setzt sich in der Software-Entwicklung zunehmend durch. Inzwischen gibt es zahlreiche Workshops, umfassende Literatur und eine große Anzahl erfahrener Scrum-Experten (Bild 1).

Im Scrum-Kontext wurde es bisher jedoch vernachlässigt, andere Elemente und fachliche Disziplinen, die essenziell für ein starkes Produkt sind, zu integrieren – wie beispielsweise das User-Experience-Design. Eine gute User Experience (UX) führt dazu, dass Anwender ein Produkt beziehungsweise eine Software gerne nutzen, und ist damit ein wesentlicher Erfolgsfaktor.

Wenige Berührungspunkte

Dennoch treten in der Praxis das User-Experience-Design und der Scrum-Prozess bisher meist nur nebeneinander an und haben wenige Berührungspunkte: Das User-Experience-Design – von User Research über die Definition von Personas bis hin zur Erarbeitung von Skizzen und Wireframes – wird oft im Voraus erstellt und dann für die Umsetzung dem Entwicklungsteam übergeben.

Erschwerend kommt hinzu, dass meist externe, spezialisierte Agenturen – die oft nach der klassischen Wasserfall-Methode arbeiten – das User-Experience-Design gestalten. Das liegt zumeist daran, dass bei der Gründung insbesondere der seit Längerem bestehenden IT-Unternehmen das Thema UX noch nicht so sehr im Vordergrund stand wie heute.

Mit der zunehmenden Bedeutung und Verbreitung des Internets – bis auf die Bildschirme unserer Smartphones – haben sich die Ansprüche der User an eine Software jedoch verändert: Sie legen deutlich mehr Wert auf das Design und die Nutzerführung einer Software oder einer Internetanwendung.

Um den stetig steigenden Ansprüchen der User gerecht zu werden und nicht zuletzt, um Marktanteile zu halten oder gar auszubauen, muss das UX-Design den User nicht nur bei der Anwendung unterstützen, sondern ihm auch ein gutes Gefühl bei der Nutzung der Software vermitteln.

Je weniger Software-Unternehmen also auf ein optimales User-Experience-Design verzichten können, umso wichtiger ist es, diese Disziplin mit dem Scrum-Prozess zu verzahnen, der von ihnen meist für die Entwicklung eingesetzt wird. Denn nur wenn das User-Experience-Design und die Entwicklung im Rahmen eines agilen Prozesses gemeinsam antreten, steht am Ende ein Sieg: ein erfolgreiches Produkt, das nicht nur funktioniert, sondern das die Nutzer auch gerne anwenden.

Mehr Austausch und gegenseitiges Verständnis

Eine gelungene agile Zusammenarbeit zwischen UX-Designern und Software-Entwicklern wird durch viele Faktoren bestimmt, der Hauptfaktor aber ist das Team selbst. Häufig ist jedoch das Verhältnis zwischen Designern und Entwicklern eher angespannt. Das liegt nicht selten daran, dass beiden Gruppen nur wenig über die Arbeitsprozesse der jeweils anderen Gruppe wissen. So ist es für Designer oft nicht leicht nachzuvollziehen, wie die Entwickler mit den von ihnen zur Verfügung gestellten Designs weiterarbeiten.

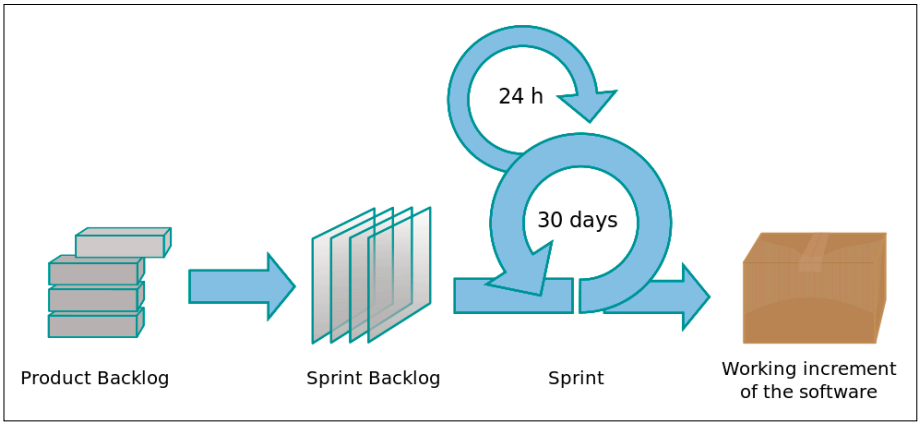


Bild: Sebastian Wallroth (Wikipedia)

Schematische Darstellung der Scrum-Projektmanagement-Methode (Bild 1)

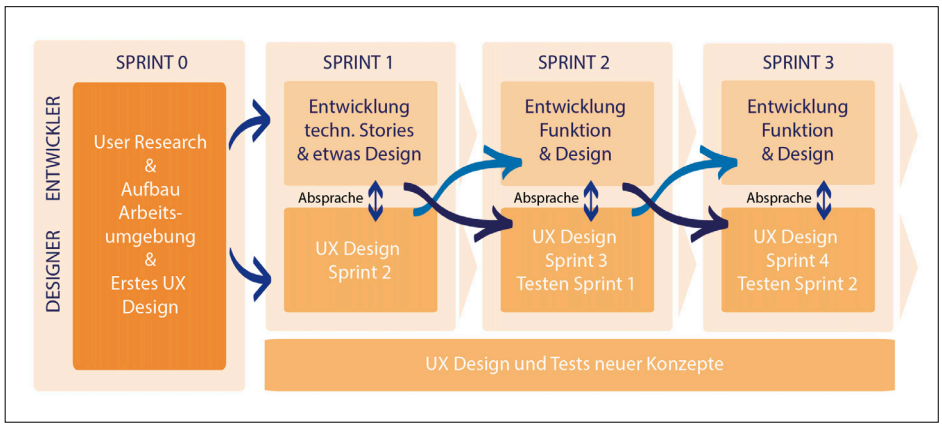
Die Entwickler wiederum können sich häufig kein Bild von den Prozessen machen, die Designer durchlaufen, um ein valides Konzept zu gestalten. Für beide Seiten ist es mitunter sehr intransparent, was die jeweils andere Gruppe in ihrer täglichen Arbeit leistet.

Integration der Designer

Aufklärungsarbeit seitens der Projektleitung und der Teammitglieder selbst kann hier zunächst ein grundlegendes Verständnis schaffen. Dabei sollte erklärt werden, was die Aufgaben der einzelnen Rollen sind und inwiefern sich durch die

Integration der Designer auch Prozesse in der Software-Entwicklung anpassen müssen. Es ist wichtig, dass alle Teammitglieder verstehen, dass nur durch ein gutes Zusammenspiel beider Prozesse – User-Experience-Design und Software-Entwicklung – ein gelungenes Produkt entstehen kann.

Um das Verständnis für die Arbeit der jeweils anderen zu verbessern, ist räumliche Nähe ein wesentlicher Aspekt: Wenn Designer und Entwickler zusammen im gleichen Raum arbeiten, bekommen sie mit, an was der jeweils andere arbeitet.



Beispiel für einen agilen Projektprozess im WaterScrum-Modell (Bild 2)

Sie können sich Fragen stellen und ihre Arbeitsschritte wie zum Beispiel Designs oder Software-Architektur-Bilder sichtbar im Raum anbringen. Durch das offensichtliche Visualisieren der aktuellen Arbeiten können einfacher und schneller Gespräche und Diskussionen entstehen.

Kurze Kommunikationswege

Diese Kommunikation – die weitaus häufiger stattfindet, wenn ein Team in einem Raum sitzt – fördert den kontinuierlichen Austausch, auch im Hinblick auf die technische Realisierung von Design-Ideen. Kurze Kommunikationswege führen dazu, dass Fragen häufiger gestellt und schneller geklärt werden, und wirken sich damit positiv auf das Produktergebnis aus.

Wenn es zeitlich möglich ist – und das entsprechende Fachwissen vorliegt –, können die Designer die Entwickler auch bei der Programmierung unterstützen, etwa im Bereich der Frontend-Entwicklung.

Der Vorteil: Selbst entworfene Designs lassen sich auch gleich technisch umsetzen und somit auf ihre Machbarkeit prüfen. Umgekehrt können sich auch die Entwickler im Rahmen des Scrum-Prozesses an der Erstellung des User-Experience-Designs beteiligen, etwa indem sie Mockups oder Prototypen erstellen.

UX-Design in Sprint 0 berücksichtigen

Im Scrum-Prozess wird die Zeit vor dem ersten Sprint für vorbereitende Maßnahmen genutzt, wie zum Beispiel die Aufstellung der technischen Architektur. Dieser Zeitraum wird auch als Sprint 0 bezeichnet (Bild 2).

In diese Phase lassen sich die ersten konzeptionellen und kreativen Prozessschritte des UX-Designs gut integrieren. Ziel von Sprint 0 ist in diesem Fall, gemeinsam ein grundlegendes Verständnis für den User und eine einheitliche Produktvision zu erarbeiten.

Hierfür empfiehlt sich ein Workshop, in dem alle Projektbeteiligten – Entwickler, User-Experience-Designer, Product Owner und idealerweise auch die Stakeholder – ein Big Picture der UX-Idee entwickeln. Ebenso sollten bereits in Sprint 0 die ersten Schritte des UX-Designs umgesetzt werden: User Research, Persona-Erstellung und das Erarbeiten von Anforderungen in Form von User Stories. Auf dieser Basis können die Beteiligten gemeinsam ein für alle verständliches Product Backlog erarbeiten. Bereits im Sprint 0 sollten Daily Scrum Meetings stattfinden, in denen sich die Teammitglieder gegenseitig über den Fortschritt und aktuellen Stand der Vorbereitungsphase informieren.

Die User Stories, die im Sprint 0 zusammen erarbeitet werden, sortiert der Product Owner am Ende des Sprints nach ihrer Priorität. Am höchsten werden die User Stories priorisiert, die das sogenannte Minimal Viable Product (MVP) abbilden – also das Produkt oder Feature, welches mit dem geringsten Aufwand den größtmöglichen Nutzen für den User bringt. Diese User Stories sollten dann in den ersten Sprints umgesetzt werden.

Aufwandsschätzung

Um abschätzen zu können, wie viele Sprints benötigt werden, um das MVP umzusetzen, sollten die Entwickler in Sprint 0 eine Aufwandsschätzung der priorisierten User Stories vornehmen. Aufgabe der User-Experience-Designer in dieser Phase ist es, den Wert und Nutzen einer User Story zu beschreiben und die Entwickler an den Aufwand für die Umsetzung des Designs zu erinnern.

Für die Entwickler kann es schwierig sein, eine Aufwandsschätzung abzugeben, wenn noch kein definiertes User-Experience-Design vorhanden ist. Um ein gemeinsames Verständnis – und damit eine genauere Schätzung – der User Stories zu erreichen, empfiehlt es sich, während Sprint 0 einfache (Papier-)Prototypen zu erstellen. Die Ideen aller Projektbeteiligten lassen sich auf diese Weise schnell visualisieren und so besser in der Gruppe diskutieren. Die tragfähigsten Ideen können dann weiterentwickelt und idealerweise anhand von Prototypen noch in Sprint 0 auf ihre Benutzerfreundlichkeit getestet werden, zum Beispiel von Kollegen. Das Feedback der Tester lässt sich dann in einer schnellen Iteration berücksichtigen.

Der parallelen Entwicklung geschuldet, wird es im Lauf des Projekts nur noch den User-Experience-Designern möglich sein, den kompletten Prozess immer wieder durchzuführen, das heißt, kontinuierliche User Research, Anpassung – und ▶

falls erforderlich Neuerstellung – von Personas, Erstellung und Test von Prototypen etc. Die User-Experience-Designer sollten im Lauf des Prozesses darauf achten, ihre gewonnenen Erkenntnisse und Resultate regelmäßig mit dem restlichen Team zu teilen.

UX-Designer an allen Scrum-Meetings beteiligen

Voraussetzung für eine ganzheitliche Integration der User-Experience-Designer in den Scrum-Prozess ist deren Teilnahme an allen Scrum-typischen Meetings. Dazu zählen Daily Scrum, Plannings, Product Backlog Refinement, Retrospektiven und Reviews.

Im Rahmen dieses institutionalisierten Austauschs zwischen Entwicklern und User-Experience-Designern – idealerweise zusätzlich unterstützt durch räumliche Nähe – können beide Gruppen schnell herausfinden, ob sich das erstellte User-Experience-Design mit angemessenem Aufwand umsetzen lässt. Weiterhin haben die Entwickler so auch die Möglichkeit, ihre Ideen und Ansichten in den UX-Design-Prozess einfließen zu lassen.

Ebenfalls empfiehlt sich eine enge Zusammenarbeit zwischen UX-Designern und Product Ownern, speziell bei der Ausgestaltung von User Stories. Sie können ihre Erfahrung im Umgang mit den Nutzern und deren Bedürfnisse bei der Erstellung von User Stories einfließen lassen und User-Experience-Designs zu den geplanten User Stories anfertigen. Dabei sollte darauf geachtet werden, sehr detailliertes User-Experience-Design erst dann zu erstellen, wenn es benötigt wird – und nicht schon weit im Voraus.

Das Gleiche gilt auch für die finale Ausformulierung der User Stories, da es sehr wahrscheinlich ist, dass sich initiale User Stories aufgrund neuer Erkenntnisse im Lauf des Projekts ändern. Somit ist es ratsam, die User Stories auch hinsichtlich ihrer User Experience erst ein bis zwei Sprints vor der geplanten Umsetzung zu finalisieren, um den Arbeitsaufwand zu minimieren.

Als fester Bestandteil eines Scrum Teams stehen die User-Experience-Designer jederzeit für Rückfragen zur Verfügung. Das trägt dazu bei, Mehrfacharbeit zu verhindern, etwa wenn ein Detail zum Verhalten des User Interfaces vom Entwickler anders verstanden und ohne Rückfrage implementiert wurde. Indem sie die Akzeptanzkriterien einer User Story prüfen, unterstützen die UX-Designer auch den Product Owner, der die User Stories maßgeblich mitprägt.

Usability-Tests durchführen

User-Tests sind ein weiteres Aufgabenfeld, dessen sich UX-Designer während der Entwicklung annehmen können. Hierfür empfiehlt es sich, mit ausgewählten Anwendern in regelmäßigen Abständen – etwa alle zwei Sprints – Usability-Tests durchzuführen, idealerweise zunächst mit Prototypen.

So hat die Zielgruppe die Möglichkeit, neu implementierte Produktinkremente zu evaluieren. Eine frühe Rücksprache mit den Anwendern hat den Vorteil, dass etwaige Änderungswünsche der User umgehend berücksichtigt werden können. Insbesondere wenn die Anpassungen bereits an Prototypen erfolgen und nicht erst an der bereits entwickelten Software, bedeutet das enorme Zeit- und Kostenersparnisse. Anpassungen lassen sich direkt in einer darauffolgenden Iteration vornehmen, oder die Änderungswünsche werden in neuen User Stories verankert.

Um den transparenten Austausch mit dem Scrum Team aufrechtzuerhalten, sollten die UX-Designer die Ergebnisse der User -Tests im Daily Scrum präsentieren.

Erfolgsfaktoren

Die drei wichtigsten Aspekte für eine erfolgreiche Integration von UX-Design in den Scrum-Prozess:

- **Erfolgsfaktor 1:** Räumliche Nähe – für mehr Austausch und gegenseitiges Verständnis.
- **Erfolgsfaktor 2:** UX-Design schon in Sprint 0 berücksichtigen – für relevante Personas und gute User Stories.
- **Erfolgsfaktor 3:** UX-Designer an allen Scrum-Meetings beteiligen – für das Big Picture.

Fazit

User-Experience-Designer können in einem Scrum-Team umfassende Aufgaben übernehmen und wesentlich zum Erfolg des Produkts beitragen. Voraussetzung dafür ist, dass die UX-Designer als Doppelpartner der Entwickler im Scrum-Prozess mitspielen.

Zum einen ist es ihre Aufgabe, nach vorne zu schauen und mittels User Research, User-Tests und Prototyp-Erstellung eine auf den Anwender ausgerichtete Entwicklung des Produkts voranzutreiben.

Zum anderen sollten sie immer für Nachfragen der Entwickler bezüglich aktuell bearbeiteter User Stories verfügbar sein, die bereits implementierten User Stories validieren und gegebenenfalls an der Zielgruppe testen.

Dabei sollten sich alle Beteiligten bewusst sein, dass es in der agilen Software-Entwicklung im Grunde keine endgültige Lösung gibt. Mit sich verändernden Bedürfnissen der User muss sich schließlich auch das Produkt verändern – indem es stetig angepasst und weiterentwickelt wird. Daher sind mit der Freigabe eines Softwareprodukts die Entwicklung und das User-Experience-Design nicht abgeschlossen. Im Prinzip beginnt in diesem Moment schon ein neues Match. ■

Heidi Oltersdorff
ist Consultant bei diva-e Digital Value Enterprise GmbH
www.diva-e.com

Impressum

Verlag
Neue Mediengesellschaft Ulm mbH
Bayerstraße 16a,
80335 München
Telefon: (089) 741 17-0,
Fax: (089) 741 17-101
(ist zugleich Anschrift aller Verantwortlichen)

Herausgeber
Dr. Günther Götz

Chefredakteur
Max Bold
– verantwortlich für den redaktionellen Teil –
E-Mail: redaktion@webundmobile.de

Schlussredaktion
Ernst Altmannshofer

Redaktionelle Mitarbeit
Philip Ackermann, Daniel Basler, Christian Bleske, Ekkehard Gentz, Thomas Hafen, Tam Hanna, Bernhard Lauer, Patrick Lobacher, Florence Maurice, Heidi Oltersdorff, Michael Rohrlich, Andreas Sachs, Jochen Schmidt, Katharina Sckommodau, Thomas Sillmann, Alexander Steireif, Markus Stäuble

Art Directorin
Maria-Luise Sailer

Grafik & Bildredaktion
Alfred Agatz, Dagmar Breitenbauch, Verena Greimel, Hedi Hefele, Manuela Keller, Simone Köhnke, Cornelia Pflanzer, Karoly Pokuta, Petra Reichenspurner, Ilka Rüther, Sebastian Scharnagl, Christian Schumacher, Nicole Üblacker, Mathias Vietmeier

Anzeigenberatung
Jens Schmidtmann, Anzeigenleiter
Klaus Ahlering, Senior Sales Manager
Telefon: (089) 741 17-125
Fax: (089) 741 17-269
E-Mail Anzeigenberatung: sales@nmg.de

Anzeigendisposition
Dr. Jürgen Bossmann
Telefon: (089) 741 17-281
Fax: (089) 741 17-269
E-Mail: sales@nmg.de

Leitung Herstellung/Vertrieb
Thomas Heydn
Telefon: (089) 741 17-111
E-Mail: thomas.heydn@nmg.de

Leserservice
Hotline: (089) 741 17-205
Fax: (089) 741 17-101
E-Mail: leserservice@nmg.de

Kooperationen
Denis Motzko
Telefon: (089) 741 17-116
E-Mail: kooperationen@nmg.de

Druck
L.N. Schaffrath Druckmedien
Marktweg 42–50
47608 Geldern

Vertrieb
Axel Springer Vertriebsservice GmbH
Objektvertriebsleitung Lothar Kosbü
Süderstraße 77
20097 Hamburg
Telefon: (040) 34724857

Bezugspreise
web & mobile developer ist das Profi-Magazin für Web- und Mobile-Entwickler und erscheint zwölfmal im Jahr. Der Bezugszeitraum für Abonnenten ist jeweils ein Jahr. Der Bezugspreis im Abonnement beträgt 76,20 Euro inklusive Versand und Mehrwertsteuer im Halbjahr, der Preis für ein Einzelheft 14,95 Euro. Der Jahresbezugspreis beträgt damit 152,40 Euro.

In Österreich sowie im übrigen Ausland kostet das Abonnement 83,70 Euro im Halbjahr. Der Jahresbezugspreis beträgt somit 167,40 Euro. In der Schweiz kostet das Abonnement 152,00 Franken im Halbjahr. Der Jahresbezugspreis in der Schweiz beträgt 304,00 Franken.

Das Abonnement verlängert sich automatisch um ein Jahr, wenn es nicht sechs Wochen vor Ablauf der Bezugszeit schriftlich beim Verlag gekündigt wird. Studenten erhalten bei Vorlage eines Nachweises einen Rabatt von 50 Prozent.

ISSN: 2194-4105

DESIGN THINKING

Innovationen produzieren

Mit Design Thinking lassen sich systematisch Innovationen produzieren.

Die digitale Transformation fordert Innovationen als einen der Erfolgstreiber. Wie aber kommt man zu Innovationen, wenn man sich nicht dem Zufall ergeben möchte? Design Thinking ist ein vielversprechender und vor allem ganzheitlicher Ansatz, der das systematische Produzieren von Innovationen möglich macht und fördert.

Design Thinking soll als Prozess des bewussten, absichtsvollen und planmäßigen Gestaltens von Objekten, Systemen oder Strukturen verstanden werden. Somit könnte man Design Thinking auch als erfinderisches Denken definieren.

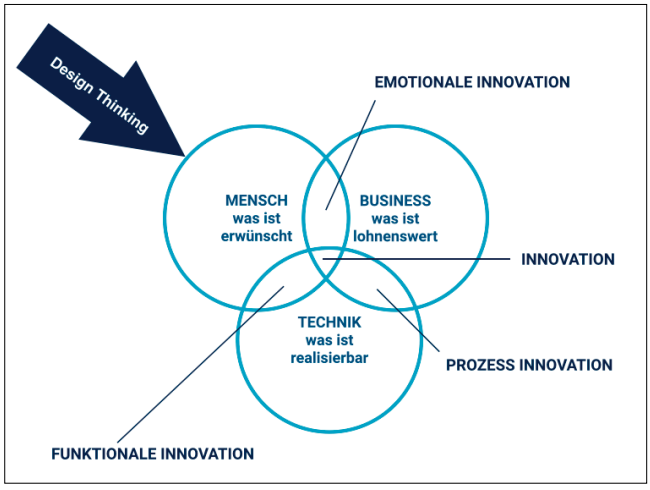
Denn sämtliche wirklich gute Ideen sind das Ergebnis von Designprozessen – manchmal als Geistesblitz und völlig zufällig, manchmal über herkömmliche Kreativmethoden (wie Brainstorming) entstanden, aber in den allermeisten Fällen steckt dort ein methodischer Ansatz dahinter.

Design Thinking ist eine Arbeitsmethodik, die verschiedene Werkzeuge miteinander verbindet, um Ideenfindung und Innovation zu fördern. Dabei ist der Ansatz völlig unabhängig von der Branche oder dem gewünschten Ergebnis.

So kann beispielsweise das Onboarding neuer Mitarbeiter, die Strategieentwicklung einer Bank oder die Entwicklung neuer Leistungen für eine Non-Profit-Organisation mit Design Thinking angegangen werden.

Die Wurzeln von Design Thinking reichen dabei zurück in die 1960er Jahre und wurden Anfang der 1990er Jahre konkreter. Dort werden die Bestrebungen der Stanford Universität in Kalifornien und der dort 1991 von David Kelly (Professor in Stanford) gegründeten Innovations-Agentur IDEO zugesprochen. Mittlerweile hat das Thema deutlich an Fahrt aufgenommen, was unter anderem am Hasso-Plattner Institut für Design (HPI) liegt, das 2005 an der Stanford Universität gegründet wurde und die dortige d.school ersetzt.

Eine Untersuchung aus dem Jahr 2014 hat zudem ergeben, dass mittlerweile die Hälfte aller deutschen DAX-Unternehmen das Potenzial von Design Thinking erkannt haben und diesen Innovationsansatz verstärkt einsetzen.



Schema: Wie Innovationen entstehen (Bild 1)

Innovation

Wörtlich bedeutet Innovation zunächst einmal: Neuerung beziehungsweise Erneuerung. Und dies ist nicht beschränkt auf besonders ausgefallene Ideen oder Erfindungen, sondern bezieht vor allem den wirtschaftlichen Erfolg eines daraus entwickelten Produkts oder einer Dienstleistung mit ein.

Damit aus einer Idee eine Innovation werden kann, muss es ein Gleichgewicht aus den folgenden Aspekten geben:

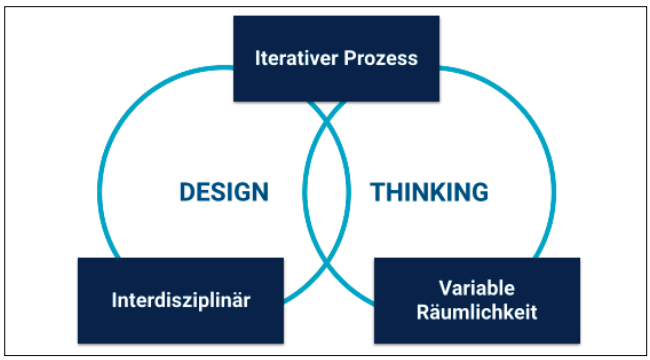
- Wünschbarkeit: Dies ist die wichtigste Komponente. Nur wenn es ein Bedürfnis und damit einen Wunsch gibt, kann Innovation entstehen. Dabei ist der Wunsch keineswegs immer offensichtlich oder kann gar artikuliert werden.
- Machbarkeit: Eine Innovation ist nur dann erfolgreich, wenn diese mit den uns gegebenen Mitteln und Möglichkeiten realisierbar ist. Sicherlich wäre ein Automat, der uns zu jedem Punkt der Erde teleportieren könnte, eine gute Idee und auch innovativ – wir werden allerdings letztlich an der Realisierbarkeit scheitern.
- Wirtschaftlichkeit: Selbst wenn fliegende Autos bald technologisch in den Bereich der Machbarkeit kommen könnten, so wäre der Preis sicherlich die größte Hürde, um mit dieser Innovation erfolgreich zu sein. Daher muss hier auf eine angemessene Wirtschaftlichkeit geachtet werden.

Richtige und nachhaltige Innovation entsteht also an der Schnittstelle zwischen menschlichen Bedürfnissen, der technologischen Machbarkeit sowie der geschäftlichen Durchführbarkeit (Bild 1). Dabei ist der Prozess des Design Thinking immer personenzentriert – das heißt, er beginnt immer mit der Frage, wann eine bestimmte Lösung für eine spezifische Zielgruppe attraktiv ist. Erst anschließend werden die Machbarkeit und die Wirtschaftlichkeit betrachtet.

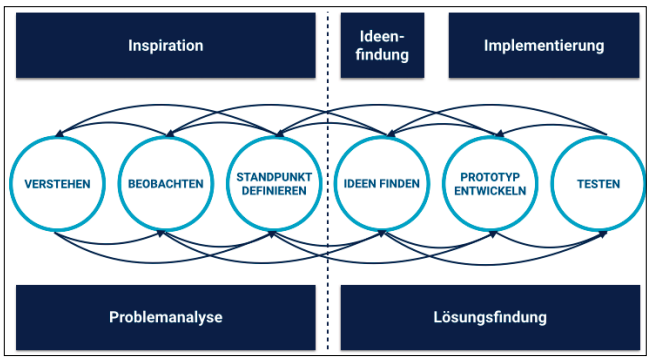
Kernelemente von Design Thinking

Maßgeblich für den Erfolg eines Design-Thinking-Projekts verantwortlich sind drei Kernelemente (Bild 2).

Oftmals entscheidet man sich bei Kreativtechniken intuitiv für homogene Teams. Diese sind eingespielt, befinden sich auf demselben intellektuellen Level, kommunizieren meist problemlos und verstehen sich blind. Aber genau das ist der größte Feind der Innovation, denn die Qualität der Ergebnis-



Die Komponenten von Design Thinking (Bild 2)



Der Design-Thinking-Prozess in der Übersicht (Bild 3)

se solcher Teams sind oft messbar schlechter beziehungsweise eben durchschnittlich. Man kennt meist schon die fertige Lösung und versäumt es, Überlegungen und Ergebnisse kritisch zu hinterfragen. Design Thinking funktioniert am besten, wenn das Projektteam aus mehreren Experten mit den unterschiedlichsten Fähigkeiten zusammengesetzt wird.

Damit kreative Prozesse in Gang gesetzt werden können, ist es hilfreich, alltägliche Strukturen aufzubrechen und eine neue Umgebung zu schaffen. Dazu bedient man sich eines offenen und variabel eingerichteten Raums, der zum Beispiel mit Hilfe von mobilen Trennwänden maximale Flexibilität bietet. Durch die Trennwände kann der Raum beliebig vergrößert oder verkleinert und damit an die jeweilige Situation angepasst werden. Weiterhin sind Pinnwände, Schreibtafeln, Stehtische und weitere mobile Tische hilfreich. Anzuraten ist ein Stuhlverbot, um bewusst die Meeting-Konventionen aufzubrechen. Die Räumlichkeiten sollen Nähe und Gemeinschaft mit Individualität in Einklang bringen – denn nur wer sich wohlfühlt, kann gute und innovative Ideen vorantreiben. Zudem sollte für genügend Material gesorgt werden.

Grundsätzlich gibt es für den Prozess keinen vorgegebenen Lösungsweg – um allerdings die Potenziale des Design-Thinking-Innovationsansatzes zu unterstützen, ist es notwendig, einen methodenübergreifenden Ansatz mit einer gewissen Systematik zu wählen. Dabei handelt es sich um sechs Prozessschritte, die iterativ miteinander verbunden sind.

Je nach Design-Thinking-Schule findet man leicht unterschiedliche Phasen für den Prozess. Die d.school und die Universität St. Gallen verwenden dafür fünf Stufen: Empathize, Define, Ideate, Prototype und Test. Das HPI beispielsweise und auch wir im Pluswerk verwenden ein 6-stufiges Modell: Verstehen, Beobachten, Standpunkt definieren, Ideen finden, Prototyp entwickeln und Testen. Alle Phasen sind iterativ miteinander verbunden. Auch wenn sich das Vorgehen bestens bewährt hat, so ist die Denk- und Arbeitskultur von Design Thinking stets offen für Erweiterungen und Ergänzungen. Daher dienen die Schritte zunächst als Orientierung (Bild 3).

Durch die Iteration ergeben sich durch wiederholte Bearbeitung und Reflexion neue Erkenntnisse, die auch als Rückkopplungseffekte bezeichnet werden. Es ist daher sehr wichtig, die Ergebnisse stets zu hinterfragen beziehungsweise zu überprüfen und gegebenenfalls mit der vorhergehenden Phase erneut abzugleichen. Neue Erkenntnisse können so bereits bestehende Ideen infrage stellen und den Prozess erneut ankurbeln. Der Arbeitsprozess verbindet grundsätzlich die analytische und intuitive Arbeitsweise.

Links zum Thema

- Harvard Business Review – Design Thinking
  http://www.ideo.com/images/uploads/thoughts/IDEO_HBR_Design_Thinking.pdf
- Hasso-Plattner-Institut Potsdam
  http://hpi.de/de/school-of-design-thinking/design-thinking.html
- Hasso-Plattner-Institute Stanford
  http://dschool.stanford.edu
- Virtual Crash Course
  http://dschool.stanford.edu/dgift
- Design Thinking Blog (IDEO)
  http://designthinking.ideo.com
- Design Thinking Workshop
  https://www.youtube.com/watch?v=o92YkEfeIbk

Fazit

Design Thinking ist eine leistungsfähige Möglichkeit, an Innovationen gezielt, methodisch und erfolgreich heranzugehen. Der Autor hat eine Vorlage entwickelt, mit der man einen Design-Thinking-Workshop bestmöglich begleiten kann. Diesen kann man sich unter dem URL www.lobacher.de/files/wmd-DT-Workshop.pdf herunterladen. ■

Patrick Lobacher
ist Digital-Native, Entwickler, Berater, Coach und Autor zahlreicher Fachbücher und Fachartikel. Er ist Vorstandsvorsitzender der Pluswerk AG, die digitale Kommunikationslösungen konzipiert, umsetzt und betreut.
http://pluswerk.ag

IOT-PROGRAMMIERUNG MIT CYLON.JS

Einfacher Zugriff

Cylon.js vereinfacht den Zugriff auf Mikrocontroller und dort angeschlossene Sensoren.

Der Zugriff auf Mikrocontroller beziehungsweise Sensoren und Aktoren wird auch im Bereich der JavaScript-Entwicklung immer interessanter: Plattformen wie Tessel (https://tessel.io) und Espruino (www.espruino.com) verfügen bereits standardmäßig über eigene JavaScript-Interpreter, sodass auf diesen Plattformen JavaScript-Anwendungen direkt ausgeführt werden können.

Für den Zugriff auf Arduino-Geräte kommt dagegen in der Regel das sogenannte Firmata-Protokoll zum Einsatz (https://www.arduino.cc/en/Reference/Firmata), wobei es auch hierfür entsprechende JavaScript-Clients gibt. Kleiner Wermutstropfen: Jeder Mikrocontroller verwendet sein eigenes SDK und definiert entsprechend sein eigenes API.

Das ist genau der Punkt, an dem Cylon.js (https://cylonjs.com) ansetzt: Dabei handelt es sich um ein JavaScript-Framework, das ein einheitliches API für verschiedene Mikrocontroller zur Verfügung stellt und das dazu verwendet werden kann, mit einer Reihe verschiedener IoT-Plattformen zu kommunizieren, zum Beispiel mit den genannten Plattformen Arduino, Espruino und Tessel. Es fügt dabei eine Abstraktionsschicht oberhalb der verschiedenen Protokolle ein, die von den unterschiedlichen IoT-Plattformen verwendet werden, und erleichtert somit den Zugriff (Bild 1).

Unterstützt werden momentan 43 verschiedene Plattformen, darunter Mikrocontroller und Mikrocomputer wie beispielsweise Arduino, Beaglebone Black, Raspberry Pi, Particle und Tessel, aber auch Drohnen wie ARDrone und Bebop oder andere IoT-Geräte wie Leap Motion, Philips Hue und die Pebble Watch.

Im Folgenden möchte ich eine kurze Einführung in Cylon.js geben und anhand verschiedener Beispiele zeigen, wie per Bluetooth Low Energy auf Informationen von Bluetooth-Geräten zugegriffen werden kann oder wie über das Firmata-Protokoll auf Komponenten von Arduino-Geräten zugegriffen werden kann.

Installation und erste Schritte

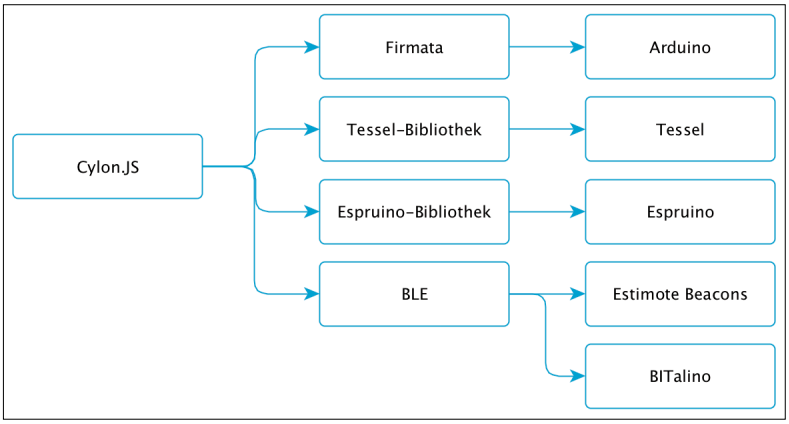
Cylon.js wird als Modul für Node.js wie gewohnt über den Befehl *npm install cylon.js* installiert. Je nachdem, welche Funktionalität man von Cylon.js verwenden möchte, ist die Installation weiterer Module notwendig (beispielsweise das Modul *cylon-firmata* als Support für das Firmata-Protokoll, das Modul *cylon-ble*, um Verbindungen über Bluetooth Low Energy aufbauen zu können, das Modul *cylon-gpio*, um auf GPIO-Pins zuzugreifen, oder *cylon-i2c* für den Zugriff auf den I2C-Bus).

Eine einfache Anwendung, die fortlaufend zweimal pro Sekunde die Meldung *Hallo Welt* auf die Konsole ausgibt, zeigt das folgende Listing:

```
'use strict';
// 1.) Importieren von Cylon.js
const Cylon = require('cylon');
// 2.) Konfiguration eines Roboters
let robot = Cylon.robot({
  work() {
    setInterval(() => {
      console.log('Hallo Welt');
    }, 500);
  }
});
// 3.) Starten des Roboters
robot.start();
```

Natürlich ist hier noch keine Verbindung zu irgendeinem Mikrocontroller oder Ähnlichem zu sehen, aber man erkennt schon den prinzipiellen Aufbau eines Cylon.js-Programms, der im Wesentlichen aus drei Schritten besteht: Importieren des Cylon.js-Moduls über *require('cylon')*, Erstellen und Konfiguration eines sogenannten Roboters (Cylon.js-Jargon für Objekte, die ein oder mehrere Geräte steuern) über die Methode *robot()* und Starten des Roboters über die Methode *start()*. Der Methode *robot()* übergibt man dabei ein Konfigurationsobjekt, dessen Kernstück die Methode *work()* ist: Diese definiert das Verhalten des Roboters.

Für das Stoppen eines Roboters gibt es übrigens nicht etwa analog zu der Methode *start()* eine Methode *stop()*. Stattdessen muss die Methode *Cylon.halt()* verwendet werden, wel-



Das Prinzip von Cylon.js (Bild 1)

Listing 1: Steuern eines BeagleBone Black

```
'use strict'
const Cylon = require('cylon');
let robot = Cylon.robot({
  connections: {
    beaglebone: {
      adaptor: 'beaglebone'
    }
  },
  devices: {
    led: {
      driver: 'led',
      pin: 'P9_12',
      connection: 'beaglebone'
    },
    button: {
      driver: 'button',
      pin: 'P9_14',
      connection: 'beaglebone'
    }
  },
  work(thisRobot) {
    thisRobot.button.on('push', () => {
      robot.led.toggle();
    });
  }
});

robot.start();
```

che global alle Roboter stoppt. Ein einfaches Beispiel hierzu zeigt das nachfolgende Listing:

```
'use strict';
const Cylon = require('cylon');
let robot = Cylon.robot({
  work() {
    let id = setInterval(() => {
      console.log('Hallo Welt');
    }, 500);
    setTimeout(() => {
      console.log('Anhalten');
      clearInterval(id);
      Cylon.halt();
    }, 5000);
  }
});

robot.start();
```

Das Beispiel von eben wurde hier dahingehend abgeändert, dass der Roboter nach fünf Sekunden angehalten wird.

Die Methode *Cylon.halt()* sorgt allerdings nicht dafür, dass intervallgesteuerte Befehle, die über *setInterval()* definiert wurden, oder zeitgesteuerte Befehle, die über *setTimeout()* definiert wurden, stoppen beziehungsweise nicht (länger) ausgeführt werden. Diese müssen – wie im Listing gezeigt – separat über entsprechende Funktionen *clearInterval()* beziehungsweise *clearTimeout()* gestoppt werden.

Konfiguration von Verbindung, Aktoren und Sensoren

So weit die Grundlagen. Jetzt stellt sich die Frage, wie man sich konkret zu einer der Plattformen verbinden und auf Komponenten, sprich auf Aktoren und Sensoren zugreifen kann. Hier kommen im Wesentlichen zwei Eigenschaften des eingangs erwähnten Konfigurationsobjekts ins Spiel: die Eigenschaft *connections* und die Eigenschaft *devices*.

Über Erstere definiert man, wie die Verbindung zu einem Gerät (oder mehreren Geräten) erfolgen soll. Über Letztere definiert man, welche Komponenten durch das Gerät beziehungsweise die Geräte bereitgestellt werden.

In Listing 1 beispielsweise wird eine Verbindung zu einem BeagleBone Black konfiguriert (https://beagleboard.org/black). Der Eigenschaft *connections* ist dabei ein weiteres Objekt hinterlegt, wobei jede Eigenschaft dieses Objekts wiederum eine Verbindung repräsentiert.

Namen können frei gewählt werden

Die Namen dieser Eigenschaften (hier *beaglebone)* können übrigens frei gewählt werden. Entscheidend ist, welcher Wert in der Eigenschaft *adaptor* hinterlegt ist. Dieser definiert, welches Cylon.js-Plug-in für den Verbindungsaufbau verwendet wird – im Beispiel das Package *cylon-beaglebone* (https://github.com/hybridgroup/cylon-beaglebone).

Über die Eigenschaft *devices* werden einzelne, durch eine Verbindung bereitgestellte Komponenten konfiguriert, sprich Aktoren und Sensoren, die durch das verbundene Gerät bereitgestellt werden.

Auch dieses Objekt enthält Eigenschaften, die jeweils die Konfiguration für einen Aktor/Sensor enthalten und prinzipiell beliebig benannt werden können (entscheidend ist der in *driver* angegebene Treibername). Im Beispiel werden eine Komponente *led* (https://cylonjs.com/documentation/drivers/led) und eine Komponente *button* (https://cylonjs.com/documentation/drivers/button) definiert. Es werden aber auch eine Reihe weiterer Komponenten unterstützt wie LCDs, Motoren und vieles mehr.

Innerhalb der Methode *work()* stehen die definierten Komponenten als Eigenschaften an dem dieser Methode übergebenen Parameter zur Verfügung (im Beispiel *thisRobot)*.

Fluent API

Alternativ zu der Konfiguration eines Roboters über geschachtelte Konfigurationsobjekte bietet Cylon.js ein Fluent API an, wodurch der Code etwas lesbarer und sprechen- ▶

Listing 2: Das Fluent API im Einsatz

```
'use strict';
const Cylon = require('cylon');

Cylon
  .robot()
  .connection(
    'beaglebone', {
      adaptor: 'beaglebone'
    })

  .device(
    'led', {
      driver: 'led',
      pin: 'P9_12',
      connection: 'beaglebone'
    })
  .device(
    'button', {
      driver: 'button',
      pin: 'P9_14',
      connection: 'beaglebone'
    })

  .on('ready', (robot) => {
  robot.button.on('push', () => {
    robot.led.toggle();
  });
});

Cylon.start();
```

der wird. Ein Beispiel dazu zeigt Listing 2: Der Code hier verwendet die Methoden *connection()* und *device()* zur Konfiguration von Verbindungen und Komponenten sowie die Methode *on()* für die Definition des Verhaltens, das vorher in der Methode *work()* hinterlegt war. Welche der beiden Varianten Sie bevorzugen, bleibt Ihnen überlassen.

Cylon.js abstrahiert also von dem Zugriff auf einzelne Geräte, ohne dass man sich als Entwickler um die Details kümmern muss. Zu sehen ist dies auch schön in Listing 3 und Listing 4. Ersteres zeigt, wie sich eine LED an einem Tessel-Gerät ein- und ausschalten lässt, Letzteres zeigt, wie sich das Gleiche für ein Arduino-Gerät bewerkstelligen lässt. Die Lo-

Listing 3: Steuern eines Tessels

```
'use strict';
const Cylon = require('cylon');
Cylon
  .robot()
  .connection(
    'tessel', {
      adaptor: 'tessel'
  })
  .device(
    'led', {
      driver: 'led',
      pin: 1,
      connection: 'tessel'
  })
  .device(
    'button', {
      driver: 'button',
      pin: 'config',
      connection: 'tessel'
  })
  .on('ready', (robot) => {
    robot.button.on('push', () => {
      robot.led.toggle();
  });
});
Cylon.start();
```

Listing 4: Steuern eines Arduinos

```
'use strict';
const Cylon = require('cylon');
Cylon
  .robot()
  .connection(
    'arduino', {
      adaptor: 'firmata',
      port: '/dev/cu.usbmodem14131' })
  .device(
    'led', {
      driver: 'led',
      pin: 13,
      connection: 'arduino'
  })
  .device(
    'button', {
      driver: 'button',
      pin: 2,
      connection: 'arduino'
  })
  .on('ready', (robot) => {
    robot.button.on('push', () => {
      robot.led.toggle();
  });
});
Cylon.start();
```

Listing 5: Zugriff über BLE mit noble

```
'use strict';
const noble = require('noble');
const SERVICE_ID = '';
const CHARACTERISTIC_ID = '';

noble.on('stateChange', (state) =>
{
  if (state === 'poweredOn')
{
  noble.startScanning();
}
else
{
  noble.stopScanning();
}
});
noble.on('discover', (peripheral) =>
{
  peripheral.connect((error) =>
  {
    peripheral.discoverServices([ SERVICE_ID
    ],
    (error, services) =>
    {
      services.forEach((service) =>
      {
        service.discoverCharacteristics
        ([ CHARACTERISTIC_ID ],
        (error, characteristics) =>
        {
          let characteristic = characteristics[0];
          characteristic.read((error, data) =>
          {
            console.log(data);
          });
        });
      });
    });
  });
});
```

gik ist in beiden Fällen exakt gleich. Das Einzige, was sich unterscheidet, sind die Konfigurationen der Verbindungen zum Gerät und zu den zur Verfügung gestellten Komponenten (in diesem Fall eine LED und ein Taster).

Mittlerweile gibt es viele verschiedene Protokolle beziehungsweise Spezifikationen, die im Bereich IoT für die Kommunikation zum Einsatz kommen, beispielsweise WiFi, ZigBee, Z-Wave oder Bluetooth Low Energy (kurz BLE). Bei Letzterem handelt es sich um eine leichtgewichtige Untermenge des klassischen Bluetooth, die Teil der Bluetooth-4.0-Spezifikation ist und von allen namhaften Plattformen (iOS, Android, Mac OS X, Windows, Linux et cetera) unterstützt wird.

Prinzipiell läuft der Zugriff per BLE in fünf Schritten ab: Scannen nach BLE-Geräten, Verbindung zu einem Gerät herstellen, Zugriff auf einen Service, der durch das Gerät zur Verfügung gestellt wird, Zugriff auf eine sogenannte Charakteristik dieses Services und Zugriff auf die der Charakteristik hinterlegten Daten. Diese fünf Schritte sind auch notwendig, wenn man das Node.js-Modul *noble* verwendet (https://github.com/sandeepmistry/noble), wie in Listing 5 zu sehen. ▶

Listing 6: Zugriff auf eine BLE-Charakteristik

```
'use strict';
const Cylon = require('cylon');
const DEVICE_UUID = '< UUID hier eingeben >';
const SERVICE_ID = '< ID hier eingeben >';
const CHARACTERISTIC_ID = '< ID hier eingeben >';

let robot = Cylon.robot({
  connections: {
    bluetooth: {
      adaptor: 'ble',
      uuid: DEVICE_UUID
    }
  },
  devices: {
    bleDevice: {
      driver: 'ble-characteristic',
      serviceId: SERVICE_ID,
      characteristicId: CHARACTERISTIC_ID,
      connection: 'bluetooth'
    }
  },
  work(thisRobot) {
    thisRobot.bleDevice.readCharacteristic(
      (error, data) => {
        if (error) {
          return console.error('Error: ', err);
        }
        console.log('Data: ', data);
      }
    );
  }
});

robot.start();
```

Listing 7: Zugriff auf generische BLE-Informationen

```
'use strict';
const Cylon = require('cylon');
const DEVICE_UUID = '< UUID hier eingeben >';

let robot = Cylon.robot({
  connections: {
    bluetooth: {
      adaptor: 'ble',
      uuid: DEVICE_UUID
    }
  },
  devices: {
    bleDevice: {
      driver: 'ble-generic-access'
    }
  },

  work(thisRobot) {
    thisRobot.bleDevice.getDeviceName
    ((error, data) => {
      if (error) {
        return console.error('Error: ', err);
      }
      console.log('Device Name: ' + data);
    });
  }
});

robot.start();
```

Cylon.js wiederum vereinfacht den Zugriff per BLE. Das entsprechende Modul *cylon-ble* basiert dabei zwar auch auf dem Modul *noble*, verbirgt aber die Komplexität vor dem Entwickler. Der entsprechende Code in Listing 6 und Listing 7 sieht daher auch schon viel lesbarer aus. Vom Prinzip her bleibt wieder alles beim Alten: erst die Konfiguration der Verbindung, anschließend die Konfiguration der Komponente (die in diesem Fall eine einzelne Charakteristik eines bestimmten BLE-Services ist), und dann Zugriff auf diese Komponente in der Methode *work()*.

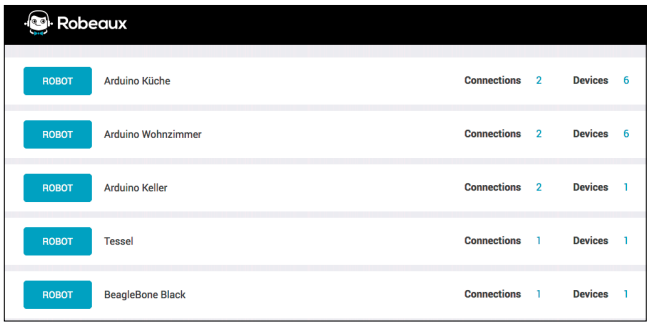
Zugriff per REST-API

Cylon.js stellt optional sogar einen kleinen Webserver inklusive eines REST-API zur Verfügung, über den sich Roboter per HTTP fernsteuern lassen. Dazu muss zunächst das Modul *cylon-api-http* wie gewohnt über NPM installiert werden: *npm install cylon-api-http*.

Anschließend startet man den Webserver über einen Aufruf von *Cylon.api('http')*, und schon steht unter *https://127.0.0.1:3000/api* ein entsprechendes REST-API zur Verfügung (Tabelle 1) und unter *https://127.0.0.1:3000* eine Weboberfläche, um auf dieses API direkt aus dem Browser heraus zuzugreifen (Bild 2). Ebenfalls erwähnenswert: Mit Hilfe von Browserify (http://browserify.org) können einzelne Cylon.js-Module auch im Browser oder in PhoneGap-Anwendungen verwendet werden (https://cylonjs.com/documentation/guides/browser-support).

Tabelle 1: Die wichtigsten Befehle des REST-API

Beschreibung	URL
Liste aller Roboter	https://127.0.0.1:3000/api/robots
Informationen zu einem Roboter	https://127.0.0.1:3000/api/robots/<robotName>
Verbindungen zu einem Roboter	https://127.0.0.1:3000/api/robots/<robotName>/connections
Informationen zu einer Verbindung	https://127.0.0.1:3000/api/robots/<name>/connections/<connectionName>
Befehle eines Roboters	https://127.0.0.1:3000/api/robots/<robotName>/commands
Befehl ausführen	https://127.0.0.1:3000/api/robots/<robotName>/commands/<commandName>



Übersicht über die verbundenen Roboter (Bild 2)

Fazit

Cylon.js erleichtert den Zugriff auf verschiedene im IoT-Bereich relevante Plattformen wie Arduino, Tessel, Espruino oder BeagleBone Black, indem es von dem konkreten Zugriff abstrahiert und eine einheitliche Schnittstelle definiert.

Auch der Zugriff auf BLE-Geräte wird durch Cylon.js erleichtert. Insgesamt eignet sich Cylon.js also sehr gut als Basis für die Entwicklung einer eigenen Middleware für das Internet der Dinge. ■

Philip Ackermann
arbeitet beim Fraunhofer-Institut für Angewandte Informationstechnologie FIT an Tools zum teilautomatisierten Testen von Web Compliance und ist Autor zweier Fachbücher über Java und JavaScript.
http://philipackermann.de

Jetzt kostenlos testen!

Das Fachmagazin für IT-Entscheider

2 Ausgaben kostenlos testen. Mit exklusivem Zugang zu unseren Digitalausgaben. Business-Newsletter inklusive.

www.com-magazin.de/gratis

DIE NEUHEITEN DER WWDC 2016

Volles Programm

Mitte Juni fand Apples alljährliche Entwicklerkonferenz in San Francisco statt.

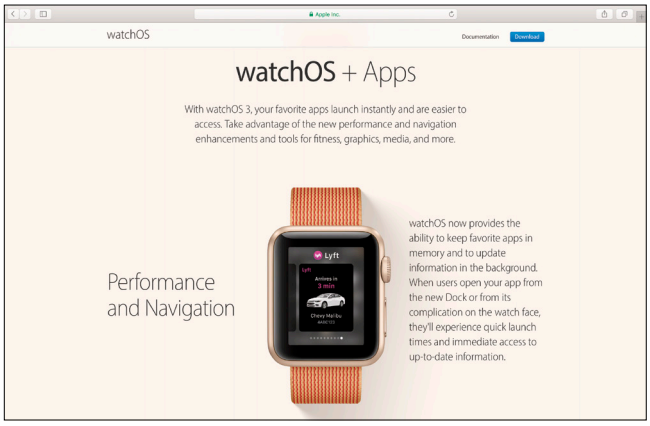
Für Apple-Entwickler ist das wohl mit Abstand die spannendste Woche des Jahres: die Woche, in der Apple auf seiner Entwicklerkonferenz WWDC über Neuheiten und Updates seiner Plattformen berichtet (Bild 1). Dabei hatte Apple in diesem Jahr ein volles Programm: Alle vier unterstützten Betriebssysteme (iOS, OS X, watchOS und tvOS) werden im Herbst dieses Jahres einen neuen Major Release erhalten und warten allesamt mit verschiedenen Neuerungen und Verbesserungen auf. Diese waren Kernthema auf der diesjährigen WWDC, neue Hardware wurde nicht vorgestellt.

watchOS 3

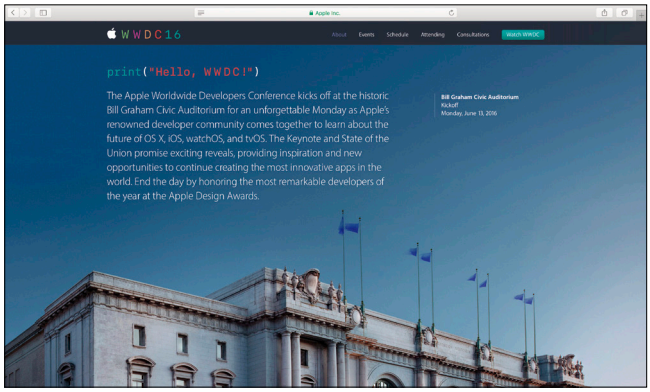
Gestartet wurde die Eröffnungs-Keynote der WWDC 2016 mit der Vorstellung der dritten Version von Apples Smartwach-Betriebssystem watchOS. Dieses läuft exklusiv auf der Apple Watch und wird im Herbst als kostenloses Update für alle bisherigen Apple-Watch-Modelle zur Verfügung stehen.

Erfreulichste und offensichtlichste Änderung in watchOS 3 ist eine deutlich gesteigerte Performance. Bereits beim Sprung von watchOS 1 auf watchOS 2 hatte Apple dank Unterstützung nativer Apps einen deutlichen Performance-Sprung hinlegen können. Mit watchOS 3 gehen sie nun aber noch einen Schritt weiter (Bild 2).

watchOS 3 ermöglicht es, Apps im Speicher zu halten und aus einem neuen Dock heraus zu starten. Anstelle eines Ladescreens, der je nach App bis zu mehreren Sekunden auf der Apple Watch angezeigt werden kann, ehe die eigentliche App geladen und zur Interaktion verfügbar ist, wachen Apps dann aus dem Speicher heraus umgehend auf und können ohne jegliche spürbare Verzögerung verwendet werden. Gerade häufig genutzte Apps werden sich damit mit watchOS 3 auf der Apple Watch insgesamt noch besser und schneller bedienen lassen, ohne regelmäßig längere Wartezeiten in Kauf nehmen zu müssen.



Die Performance von Apple-Watch-Apps nimmt mit watchOS 3 noch einmal zu (Bild 2)



Die WWDC gehört für Apple-Entwickler zu den spannendsten Wochen des Jahres (Bild 1)

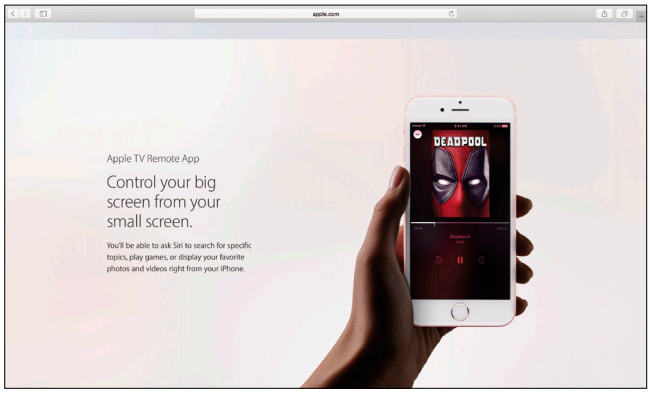
Öffnung für den Fitness-Bereich

Performance ist aber nur ein Thema in watchOS 3. Der Bereich Fitness, der bereits jetzt einen Großteil der Apple-Watch-Funktionalität ausmacht, wird ebenfalls weiter ausgebaut. Besonders spannend ist dabei die Öffnung weiterer APIs für Dritt-Entwickler, um Fitness- und Workout-Apps noch mehr Möglichkeiten und Zugriff auf weitere Informationen (beispielsweise die Herzfrequenz in Echtzeit) zu geben. Auch können Fitness-Apps bei einem Workout nun im Hintergrund weiterlaufen.

Auch finden weitere aus den anderen Plattformen bekannte Frameworks und Funktionen Einzug in watchOS 3. So sind mit SpriteKit und SceneKit nun auch Spiele mit Apples bekannten Gaming-Frameworks auf der Apple Watch möglich, auch Game Center wird mit Version 3 von watchOS unterstützt. Dank CloudKit können nun auch Apps von Dritt-Entwicklern direkt von der Apple Watch auf die iCloud zugreifen, ohne dazu den Umweg über die zugrunde liegende iPhone-App gehen zu müssen. Auch Apple Pay steht nun in eigenen Apps zur Verfügung, ebenso wie Funktionen des HomeKit-Frameworks zur Heimautomatisierung. Und mit der neuen sogenannten Face Gallery ist es erstmals möglich, eine Übersicht über alle durch Apps auf der Apple Watch installierten Komplikationen einzusehen.

tvOS 10

Mit Version 10 bekommt das erst im vergangenen Jahr erstmals vorgestellte tvOS sein erstes großes Update. Die Versi-



Eine neue Remote-App für das iPhone soll eine vereinfachte Bedienung des Apple TV erlauben (Bild 3)

onsnummer mag dabei verwirren. Offensichtlich orientiert sich tvOS aber an der Versionsnummer von iOS und startete entsprechend auch im vergangenen Jahr bereits mit der Versionsnummer 9.

Große Änderungen hat tvOS 10 trotz des großen Versionssprungs nicht zu bieten. Wie unter watchOS 3 auch wird das API weiter geöffnet und es stehen einige aus den anderen Plattformen bekannte Frameworks und Funktionen nun auch auf dem Apple TV zur Verfügung. Auch wird es zum Start von tvOS 10 im Herbst eine passende neue Remote-App für das iPhone geben, die das bequeme Steuern des Apple TV über das Smartphone ermöglichen wird (Bild 3).

Die größten Optimierungen erfährt tvOS 10 unter der Haube, im Bereich Privatsphäre und Sicherheit. Dabei geht es vor allen Dingen um die Unterstützung aktueller Sicherheitsstandards, die auch in eigenen Apps genutzt werden können.

Ähnlich wie in watchOS 3 findet auch HomeKit Einzug in tvOS, womit es möglich wird, die Heimautomatisierung auch über das heimische Apple TV durchzuführen. Das Apple TV fungiert im Idealfall somit als zentraler Hub, über den das gesamte Zuhause komfortabel gesteuert werden kann. Entwickler können diese Schnittstelle entsprechend nutzen, um eigene Apps für die Heimautomatisierung für das neue Apple TV anzubieten.

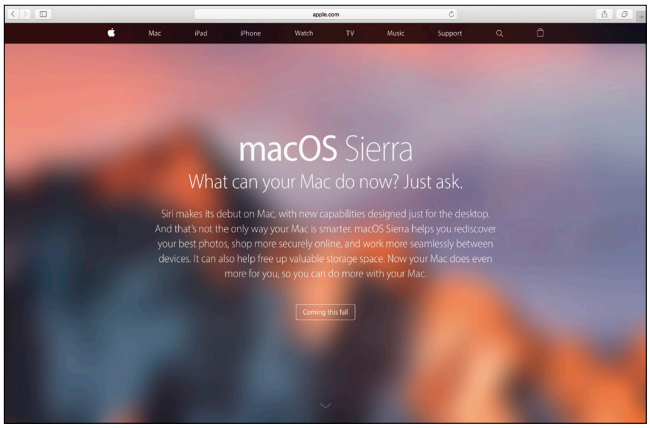
Wie auch für watchOS 3 existiert eine Beta für tvOS 10, die registrierte Apple-Entwickler auf Apples Developer Portal kostenfrei herunterladen können. Der finale Release von tvOS 10 ist für den Herbst dieses Jahres geplant, das Update wird dann kostenlos für das neue Apple TV mit tvOS 9 zur Verfügung stehen.

macOS Sierra

Im Vorfeld gab es bereits einige Spekulationen über die neue Version von Apples Mac-Betriebssystem, vorrangig darüber, was dessen Namen angehen wird. OS X passte einfach nicht länger so recht in Apples sonstige Namensschemas, die alle mit »OS« enden. Wenig überraschend gab Apple somit auch zu Beginn der Präsentation der neuen Mac-Betriebssystemversion bekannt, dass dieses fortan unter dem Namen macOS vertrieben wird (Bild 4). Darüber hinaus behält das Mac-Betriebssystem aber auch den Zusatznamen bei (wie bisher beispielsweise Yosemite oder El Capitan). Version 10.12 von macOS wird auf den Namen Sierra hören.

Eine der größten Neuerungen von macOS Sierra ist Siri. Apples Sprachassistentin schafft es mit dem kommenden Update nun auch auf den Mac. Wie bereits von iOS bekannt, können so Sprachbefehle gegeben und ausgeführt werden. So kann Siri auf dem Mac beispielsweise beim Suchen bestimmter Dateien helfen, indem man einen Zeitraum und ein Thema vorgibt. Auch Erinnerungen können mit Siri erstellt und gespeichert werden, ebenso ist das Versenden von Nachrichten möglich. Überhaupt wird damit eine großräumige Steuerung des Mac mittels Sprache durchführbar. Ob Anzeigen von Fotos, Erstellen und Ändern von Terminen, Abspielen von Musik, die Frage nach dem Wetter oder auch nach dem verbleibenden Speicherplatz auf dem Mac werden mit Siri in macOS Sierra Wirklichkeit.

Auch Apple Pay wird auf dem Mac nutzbar, zumindest indirekt. Apple wird eine Schnittstelle bieten, über die Websites es ermöglichen können, eine Bezahltransaktion über Apple Pay durchzuführen. Zur Authentifizierung muss der Nutzer dann lediglich die Zahlung über sein iPhone autori-



OS X heißt jetzt macOS, die neue Version hört auf den Namen Sierra (Bild 4)

sieren, das war's. Zum jetzigen Zeitpunkt bleibt jedoch abzuwarten, ob sich diese Schnittstelle auf vielen Websites durchsetzen wird oder eher eine Nischen- und Insellösung bleibt.

Der Mac erhält darüber hinaus ein weiteres nettes Feature, das Besitzer eines iPad Pro bereits kennen. Die Rede ist von einer neuen Bild-in-Bild-Funktion. Videos, die beispielsweise im Browser laufen, lassen sich verkleinern und an einer beliebigen Stelle auf dem Bildschirm platzieren. Während das Video in dieser Form weiterläuft, kann der Nutzer parallel den Mac wie gewohnt bedienen und Anwendungen benutzen. Dabei bleibt das Video stets sichtbar.

Optimierung von Speicherplatz und Apps

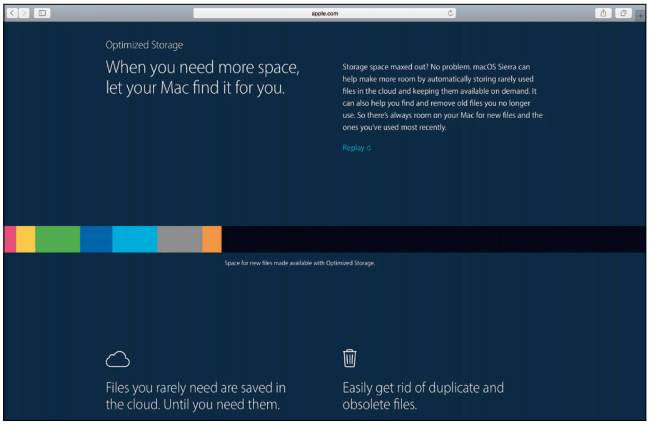
Mit einer neuen Funktion wird es in macOS Sierra möglich, den vorhandenen Speicherplatz zu optimieren und möglicherweise nicht länger benötigte Dateien auszumachen und zu entfernen. Dazu gehören beispielsweise heruntergela- ▶

dene Filme und Serien in iTunes, die schon lange nicht mehr gesehen wurden, sowie auch Cache-Dateien oder veraltete Backups von iOS-Geräten (Bild 5).

Ein neuer Assistent erlaubt es, derartige Dateien aufzuspüren und auf Wunsch zu entfernen, um so weiteren Speicherplatz auf dem Mac zur Verfügung zu stellen. Wenn das Tool wie versprochen arbeitet, ist es sicherlich eine interessante Ergänzung, nicht zuletzt, da sich bei vielen Macs Speicherplatz entweder gar nicht oder nur mit viel Aufwand austauschen und erweitern lässt.

Ebenfalls gelungen an dem genanntem Feature: Wird ad hoc freier Speicherplatz gebraucht, kann macOS Sierra auch aktuell nicht benötigte und verwendete Dateien in die iCloud auslagern, um so umgehend ein wenig Freiraum lokal auf dem Mac zu schaffen.

Neben all diesen Verbesserungen und Optimierungen in macOS Sierra werden auch viele der vorinstallierten System-



Ein neues Tool ermöglicht unter macOS Sierra das Aufräumen von Speicherplatz (Bild 5)

Apps überarbeitet und erweitert. Beispielsweise wird es mit der neuen Fotos-App möglich sein, neue von Apple generierte Zusammenstellungen der eigenen Bilder zu betrachten (sogar inklusive dramatisch gestalteten Videos). Und auch iTunes wird weiter optimiert, insbesondere um die Verwendung von Apple Music einfacher und übersichtlicher zu gestalten.

iOS 10

Zum Abschluss der diesjährigen WWDC stellte Apple den neuen Major Release seines aktuell wohl wichtigsten Betriebssystems vor. In diesem Herbst erscheint iOS 10, das mit vielen Neuerungen sowohl für Endkunden als auch für Entwickler begeistern will.

Dabei stellt das größte Highlight für Entwickler sicherlich die Öffnung vieler neuer APIs dar, mit denen es erstmals möglich wird, Erweiterungen (sogenannte Extensions) für weitere verschiedene System-Apps zu entwickeln. So sind mit iOS 10 Extensions für die Karten-App möglich. Eine App, die beispielsweise Buchungen bei Restaurants anbietet, könnte nun als Extension direkt aus der Karten-App heraus aufgerufen werden, um darüber eine entsprechende Buchung zu tätigen. Oder das Rufen eines Taxis vom aktuellen Standort könnte mit Hilfe einer passenden Extension in der Karten-App umgesetzt werden. Apps, die ortsbezogene Services nutzen, erhalten hier unter Umständen spannende neue Möglichkeiten, um den Nutzer auch außerhalb der eigentlichen App mittels Extension zu begeistern.

Ebenso öffnet sich die nativ installierte Telefon-App für Dritt-Entwickler. Apps, die VoIP für Gespräche verwenden, können nun mit Hilfe des neuen CallKit-Frameworks direkt aus der Telefon-App heraus genutzt werden. Für den Nutzer hat ein solches Verhalten den großen Vorteil, das er für einen Anruf nicht zwischen verschiedenen Apps und Messengern wechseln und auswählen muss, sondern stattdessen direkt aus der Telefon-App heraus die gewünschte App wählt, ohne die Telefon-App selbst zu verlassen.

Auch hier sind Extensions wieder das Zauberwort, denn darüber wird dieses Verhalten technisch in Apps von Dritt-Entwicklern umgesetzt.

Mit am spannendsten dürfte aber eine neue Extension-Möglichkeit für Apples Sprachassistentin Siri sein (Bild 6). Mit Apps für iOS 10 wird es erstmals möglich, die eigene App mittels Siri anzusprechen und darüber Befehle auszuführen. Herzstück des Ganzen ist das neue SiriKit-Framework. Selbst wenn die eigene App nicht aktiv ist, kann Siri mit Hilfe passender Befehle Aktionen ausführen und somit die eigene App aktivieren und nutzen. Damit schafft Apple mannigfaltige neue Möglichkeiten für App-Entwickler, die sich sehr wahrscheinlich sehr schnell in sehr vielen Apps finden werden.

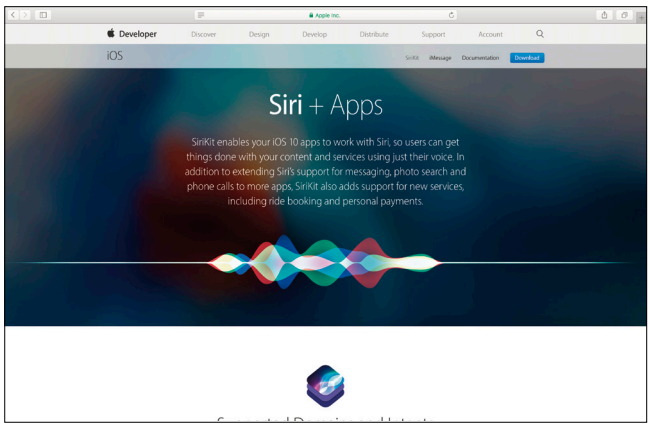
Xcode 8 und Swift 3

Zusammen mit all den neu angekündigten Versionen seiner vier Plattformen hat Apple auch die neue Version 8 seiner Entwicklungsumgebung Xcode auf der WWDC 2016 vorgestellt. Sie steht als Beta zum Download für registrierte Apple-Entwickler bereit und wird im Herbst zusammen mit den anderen Betriebssystem-Updates offiziell erscheinen.

Xcode 8 hat sich dabei primär unter der Haube verändert. Es kann nun besser mit Fehlern umgehen, bietet neue Fix-its für bestimmte Problemfälle an und soll stabiler und performanter als der Vorgänger sein. Auch enthält es natürlich alle neuen SDKs für iOS, macOS, watchOS und tvOS und unterstützt darüber hinaus die neue Version 3 von Apples Programmiersprache Swift.

Apropos Swift: Die kommende Version 3 stellt den ersten Major Release der Sprache dar, seit sie Open Source ist, und Apple hat stark betont, dass die Community einen immensen Beitrag zur Weiterentwicklung der Sprache beigetragen hat. Dabei soll die kommende Version 3 von Swift auch die letzte sein, die vorherigen Code in großen Teilen unbrauchbar macht.

Aufgrund neuer API Design Guidelines in Swift 3, die Apple selbst unter anderem auch bei den Cocoa- und Cocoa-Touch-Frameworks berücksichtigt hat, ändern sich immens viele Aufrufe, ohne die allgemeine Funktionalität und den Aufbau der Programmiersprache zu verändern. Entwickler müssen mit Version 3 also nicht komplett umlernen, sondern



Erstmals können mit iOS 10 Apps von Dritt-Entwicklern ebenfalls Siri nutzen (Bild 6)

vielmehr den neuen API Design Guidelines gerecht werden. Erfreulicherweise hilft an dieser Stelle ein Migrationsassistent in Xcode 8, von dem aus bestehende Projekte für Swift 3 vorbereitet werden können.

Als Zwischenversion erscheint vor Swift 3 noch Swift 2.3. Diese ist noch voll funktionsfähig mit Swift-2-Code, gibt aber bereits Warnungen an Stellen aus, die spätestens mit Swift 3 nicht mehr funktionieren werden. Swift 2.3 erlaubt somit einen sanften Übergang für Entwickler zu Swift 3, ohne dass direkt ein gesamtes Projekt möglicherweise nicht mehr funktionsfähig ist.

Swift Playgrounds für iPad

Zum Abschluss der Eröffnungskeynote der WWDC 2016 präsentierte Apple dann noch ein echtes Schmankerl für all jene, die auch auf dem iPad programmieren möchten. Mit der neuen App Swift Playgrounds wird nämlich genau das möglich.

Swift Playgrounds erlaubt das Erstellen der bereits aus Xcode bekannten Playgrounds auch auf dem iPad. Das geht sogar so weit, dass Xcode-Playgrounds (solange sie auf iOS basierende Frameworks verwenden) auch in Swift Playgrounds auf dem iPad verwendet und ausgeführt werden können. Mittels iCloud Drive können und sollen so Playgrounds zwischen Mac und iPad ausgetauscht und die jeweils aktuellste Version eines Playgrounds vorgehalten werden.

Um das Programmieren zu vereinfachen, verfügt Swift Playgrounds über eine entsprechend überarbeitete Bildschirmtastatur, die auf die Bedürfnisse von Programmierern ausgerichtet ist. Sie enthält auch eine kontextabhängige Direktauswahl für passende nächste Befehle, um schnelles Schreiben von Code zu ermöglichen. Zusätzliche Elemente wie ein Farb- oder Zahlenpicker sollen die Arbeit mit der App vereinfachen und das Programmieren auf optimale Art und Weise auf das iPad bringen.

Neben dem Programmieraspekt besitzt Swift Playgrounds aber noch eine weitere wichtige Funktion: Mit Hilfe von von Apple konzipierten kleinen Lernprogrammen, die in Swift Playgrounds enthalten sind, sollen vor allem junge Menschen das Programmieren lernen und Spaß daran finden. Die Lerneinheiten sind witzig und nett aufgebaut und handeln davon, ein Wesen erfolgreich durch kleine Level zu navigieren.

Apple möchte diese Funktionalität als Basis für angehende Entwickler verwenden und damit gerade an Schulen Programmierkenntnisse vermitteln. Dazu hat Apple auch neue Bücher im hauseigenen iBooks-Store bereitgestellt, die sich intensiv mit dem Lernen der Programmierung gerade auch an Schulen im Detail beschäftigen.

Swift Playgrounds ist Teil der iOS 10 Beta und kann darüber bereits getestet werden. Die finale Version erscheint dann im Herbst (voraussichtlich zusammen mit iOS 10) und wird kostenlos zum Download zur Verfügung stehen.

Fazit

Apples diesjährige Entwicklerkonferenz stand ganz im Zeichen der Software. Alle vier von Apple gepflegten Plattformen erhalten diesen Herbst ein Update und wurden im Detail auf der WWDC präsentiert. Alle Plattformen erhalten sinnvolle und passende Ergänzungen und entwickeln sich gut weiter. Insbesondere das Zusammenspiel der einzelnen Geräte untereinander nimmt immer weiter zu und wird ganz offensichtlich auch zu einem immer wichtigeren Punkt für Apple. Kein Wunder, lassen sich dadurch doch womöglich Käufer von Produkten der einen Plattform auch für eine oder mehrere andere begeistern. Schönes Beispiel dabei ist die Möglichkeit, den eigenen Mac mittels Apple Watch aus dem Stand-by beziehungsweise Ruhezustand heraus zu entsperren, ohne das Passwort eingeben zu müssen. Oder die Möglichkeit, auf dem Mac aus dem Browser heraus Apple Pay zu nutzen und die Zahlung dann einfach und bequem mittels iPhone zu authentifizieren.

Daneben zeigte Apple auch einmal mehr, wie wichtig ihnen Schulen und Bildungseinrichtungen sind. Mit Swift Playgrounds haben sie eine sehr mächtige neue App im Portfolio, die es erstmals erlaubt, Swift-Code auf einem iPad zu schreiben und direkt auszuführen. Auch wenn der Funktionsumfang nicht einmal ansatzweise an Apples vollwertige Xcode-IDE heranreicht, so lässt dieser Schritt bereits nichtsdestoweniger erahnen, wohin die Reise langfristig möglicherweise gehen wird.

Nun sind wieder wir Entwickler gefragt, uns mit den neuen Betriebssystemversionen und APIs auseinanderzusetzen und – wenn angebracht – die eigenen Apps mit neuen Features zu optimieren. Gerade iOS bietet da mit seinen vielen neuen Extensions eine große Anzahl an Möglichkeiten, um die eigenen Apps noch besser zu machen. ■

Thomas Sillmann
ist iOS-App-Entwickler, Trainer und Autor. Freiberuflich tätig programmiert er für den App Store eigene Apps sowie Apps in Form von Kundenaufträgen. Er ist Autor eines erfolgreichen Fachbuchs und mehrerer Artikel in Fachzeitschriften.
www.thomassillmann.de

DATENSCHUTZANFORDERUNGEN BEI DER ENTWICKLUNG MOBILER ANWENDUNGEN

Datenschutz und Apps

Das Thema Datenschutz muss bei der Entwicklung von Apps sorgfältig beachtet werden.

Bei der Entwicklung einer App müssen viele Herausforderungen wie Spezifikationen, Implementierung und Tests gemeistert werden, die mit dem Release ihren Höhepunkt finden.

Ein Thema, dass meist stiefmütterlich behandelt, wenn nicht gar vergessen wird, ist der Datenschutz. Dabei ist dies ein Themengebiet, mit dem jeder (mehr oder weniger) privat zu tun hat: Die Snowden-Enthüllungen haben schließlich eine umfassende Überwachung des Internets zum Vorschein gebracht, große amerikanische IT-Konzerne verarbeiten die eigenen personenbezogenen Daten mit sozialen Netzwerken sowie Kommunikationsanwendungen, und das Online-Tracking lässt beim genauen Nachdenken die Anonymität des Internets mehr als fragwürdig erscheinen.

Das Thema Datenschutz spielt auch bei der Entwicklung einer App eine wichtige Rolle, da es gesetzliche Anforderungen einzuhalten gilt. Diese werden im Folgenden näher beleuchtet.

Foto: Shutterstock / aurielaki

Personenbezogene Daten

Beim Datenschutz geht es um personenbezogene Daten. Dies sind Einzelinformationen, die einer natürlichen Person zugeordnet werden können. Im einfachsten Fall ist damit ein Name, eine Anschrift, die Bankverbindung oder eine Kundennummer gemeint.

Aber damit nicht genug: Auch Informationen, die mittelbar, also mit Zusatzwissen, zu einer Person führen können, werden zu den personenbezogenen Daten gezählt. Dazu gehören zum Beispiel ein Foto, ein Arztbericht, Bewegungsprofile oder die IP-Adresse.

Der Personenbezug der IP-Adresse ist ein Diskussionsfeld, das seit Jahren intensiv geführt wird. Die Vertreter der sogenannten Lehre des relativen Personenbezugs sind der Meinung, dass eine IP-Adresse für einen Betreiber eines Servers kein personenbezogenes Datum darstellt, da dieser unmittelbar keine Möglichkeit besitzt, von dieser Adresse auf eine konkrete Person zu kommen.

Die Vertreter des absoluten Personenbezugs, zu denen auch die Datenschutzaufsichtsbehörden zählen, sind dagegen der Auffassung, dass schon die Möglichkeit einer Verknüpfung mit anderen Daten, die juristisch ausgedrückt vernünftigerweise zur Verfügung stehen, ausreichend ist, um eine IP-Adresse als personenbezogenes Datum anzusehen.

Dieser Grundsatzstreit ist mittlerweile beim europäischen Gerichtshof gelandet und sollte noch im Jahr 2016 entschieden sein. Es ist sehr empfehlenswert, die IP-Adresse bei der Entwicklung einer App als personenbezogenes Datum anzusehen, um die damit verbundenen datenschutzrechtlichen Anforderungen rechtzeitig und damit kostengünstig umsetzbar zu machen.

Ein weiteres Feld, das sich einem nicht unmittelbar erschließt, sind die eindeutigen Gerätekennungen eines mobilen Endgeräts. So wird von den Datenschutzaufsichtsbehörden beispielsweise die IMEI-Nummer oder die MAC-Adresse der Netzwerkkarte eines Smartphones als personenbezogenes Datum angesehen.

Der Gedankengang dahinter ist, dass andere Apps oder der Betreiber der mobilen Plattform, zum Beispiel Google, ebenfalls die eindeutige Gerätekennung auslesen und möglicherweise mit einer E-Mail-Adresse, einer Kreditkarte oder einem Namen verknüpfen könnten.

In der Struktur des europäischen Datenschutzrechts, zu dem auch das deutsche Bundesdatenschutzgesetz (BDSG) gehört, gibt es für die Zulässigkeit einer Verarbeitung personenbezogener Daten einen fundamentalen Grundsatz: Entweder wird die Person um Erlaubnis gefragt, oder es gibt eine andere Rechtsgrundlage für die Verarbeitung.

Dieser Grundsatz wird durch die Datenschutzgesetze mit Leben erfüllt, die hauptsächlich die Ausnahmen beschreiben, wie ohne Erlaubnis, das heißt Einwilligung, personenbezogene Daten verarbeitet werden dürfen.

Die Einhaltung der Datenschutzgesetze überwachen die Datenschutzaufsichtsbehörden. Im Unterschied zum Verbraucherschutz sehen die Datenschutzaufsichtsbehörden auch die Unternehmen, man sagt auch »Verantwortliche Stellen«, als Grundrechtsträger (mit dem Recht auf freie Berufsausübung) an und versuchen (mehr oder weniger erfolgreich), die Interessen der Bürger und der Unternehmen in Einklang zu bringen.

Bei der Entwicklung von Apps spielt ein weiteres Gesetz eine bedeutende Rolle, das Telemediengesetz (TMG). Dieses wurde mit dem Anspruch entworfen, den Einsatz von Internetdiensten auf eine verlässliche und klare Grundlage zu stellen – geklappt hat dies mit eher fragwürdigem Erfolg.

Datenschutzerklärung

Ein wichtiger, wenn auch erst einmal nicht sehr technischer Punkt ist die Datenschutzerklärung einer App (Bild 1). Diese soll es einem Kunden zum einen ermöglichen, sich vor der Nutzung der App über die Verarbeitung der eigenen personenbezogenen Daten zu informieren.

Dies erfolgt im App-Store bei der jeweiligen App über einen Link auf die eigene Webseite. Der Link muss dann zu einer Datenschutzerklärung führen, die spezifisch für die App ist.

Ein Verweis auf die allgemeine Datenschutzerklärung der Webseite wäre nicht ausreichend. Wichtig ist zu wissen, dass eine Verlinkung nach deutschen Datenschutzgesetzen verpflichtend ist – auch wenn die App-Stores hier nur die Möglichkeit dazu anbieten, das heißt, keine Verpflichtung darin sehen. Der Gesetzgeber hat auch Bußgelder vorgesehen, sollte eine Datenschutzerklärung nicht vorhanden sein.

Die nächste Frage, die sich einem stellen könnte, wäre, was denn in die Datenschutzerklärung hineingehört. Auch wenn es verführerisch erscheint, ist von einem Copy and Paste einer anderen oder generischen Erklärung dringend abzuraten. Im Prinzip ist der Inhalt gar nicht so schwer: Es muss beschrieben werden, wer personenbezogene Daten erhält, wie sich diese zusammensetzen, was mit diesen Daten gemacht wird und an wen diese weitergegeben werden.

Dies nun ein wenig genauer: Bei den Datenempfängern sind die verschiedenen Akteure bei der Entwicklung einer App zu klären. Der (aus Sicht des Datenschutzes) wichtigste Akteur ist das Unternehmen, das über die Verarbeitung der personenbezogenen Daten grundsätzlich entscheidet, da daran die sogenannte Verantwortliche Stelle festgemacht wird. Dies ist im Allgemeinen das Unternehmen, das eine App im App-Store anbietet.

Verlinkung der Datenschutzerklärung im App Store (Google Play) (Bild 1)

Entwickelt ein externer Dienstleister die App, dann hat dieser meist keine direkte Verantwortlichkeit bezüglich der Einhaltung der Datenschutzgesetze. Dies hört sich allerdings besser an, als es ist, da ein Auftraggeber sicherlich sehr unerfreut reagiert, wenn dieser ein Bußgeld für seine App bezahlten muss, dessen Ursache ein Softwarehaus (mit) zu vertreten hat.

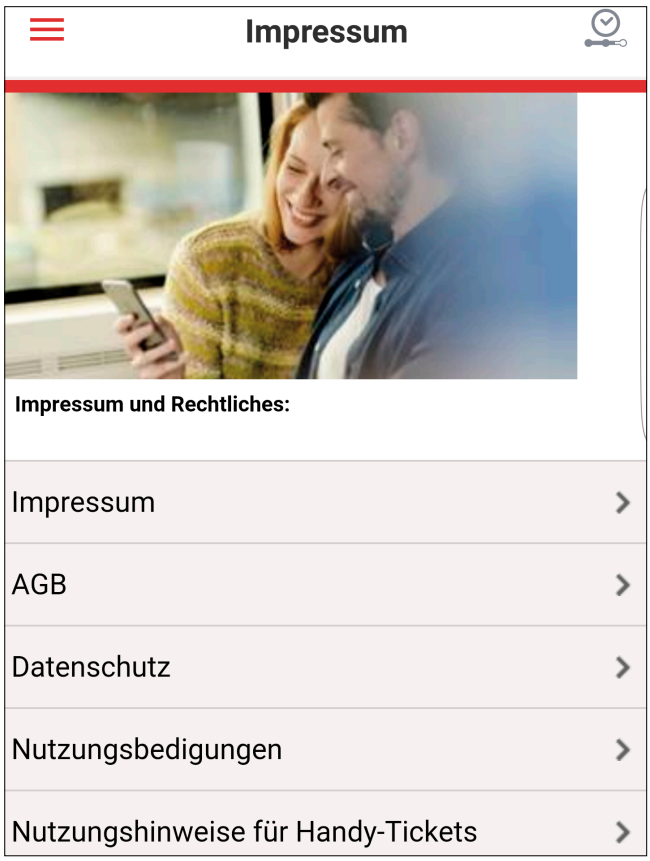
In der Datenschutzerklärung muss die verantwortliche Stelle angegeben werden, idealerweise mit einer Kontaktmöglichkeit (zum Beispiel per E-Mail), damit der Kunde sich bei Fragen oder Beschwerden zum Datenschutz an das Unternehmen werden kann.

Es muss auch eine detaillierte Beschreibung der personenbezogenen Daten erfolgen, die mit der App verarbeitet und über das Internet an ein Backend-System übertragen werden. In diesem Zusammenhang ist es auch notwendig, die Verwendung von App-Berechtigungen konkret für die App zu erläutern. Beispielsweise muss beschrieben werden, wofür ein Zugriff auf die Kontakte erfolgt, welche Datenfelder (Name, Adresse, Telefonnummer …) zu welchem Zweck ausgelesen und an wen übermittelt werden.

Auch die Datenverarbeitung auf Servern im Backend muss beschrieben werden. In diesem Zusammenhang ist es zum Beispiel wichtig, darüber zu informieren, welche Cloud-Services verwendet werden und in welchem Land der Cloud-Betreiber (und nicht die Server) seinen Sitz hat.

Sollten die Daten von der App oder vom Backend an andere Unternehmen weitergegeben werden, dann gilt es aufzupassen. Obwohl manche Szenarien rechtlich gut abbildbar sind (zum Beispiel Zahlungsdienstleister) könnte nun der Zeitpunkt sein, sich Beratungsleistungen zum Datenschutz zu holen – oder einfach mal bei der zuständigen Aufsichtsbehörde anzurufen.

Wurde die Datenschutzerklärung in leicht verständlichen Worten erstellt und für die Verlinkung im App-Store vorbereitet, muss deren zweite Verwendung geplant werden. Die Datenschutzerklärung muss nämlich ebenfalls jederzeit aus der App heraus aufrufbar und leicht auffindbar sein. Empfehlenswert ist es, in einem Einstellungsmenü einen eigenen ▶



Datenschutzerklärung innerhalb der App: Leicht auffindbar und klar strukturiert (Bild 2)

Punkt *Datenschutz* oder *Rechtliche Hinweise* zu haben, der dann zu den Inhalten der Datenschutzerklärung führt (Bild 2).

Anforderungen an die Informationssicherheit

Werden personenbezogene Daten verarbeitet, sind auch Schutzmaßnahmen gegen unbefugten Zugriff auf diese Daten umzusetzen. Im Allgemeinen sagt man, dass die Schutzziele Vertraulichkeit, Verfügbarkeit und Integrität sichergestellt werden müssen:

- Vertraulichkeit: Es muss mit Maßnahmen sichergestellt werden, dass keine Unbefugten auf die personenbezogenen Daten zugreifen können. Neben Cyberkriminellen, Hackern und Geheimdiensten darf auch der einzelne Innentäter im Unternehmen nicht vergessen werden.
- Verfügbarkeit: Die Daten müssen dann vorhanden sein, wenn diese benötigt werden. Damit sind zum Beispiel getestete Backup-Konzepte, Maßnahmen gegen Denial-of-Service-Attacken und ein sicherer Rechenzentrumsbetrieb gemeint
- Integrität: Die (unerkannte) Unveränderlichkeit der Daten muss sichergestellt werden. Ursachen können Angriffe, aber auch Programmierfehler sein. Mittel der Wahl sind kryptografische Verfahren wie Signaturen oder Hashfunktionen.

Maßnahme Nummer eins ist die Verschlüsselung personenbezogener Daten. Klar abzuraten ist von Eigenlösungen, die scheinbar hohe Sicherheit versprechen, kennt doch keiner die Ideen der eigenen Implementierung. Stattdessen fordert der Gesetzgeber, dass der Stand der Technik einzuhalten ist – die Beschäftigung mit (angewandter) Kryptografie gehört damit zum Anforderungsprofil eines App-Entwicklers.

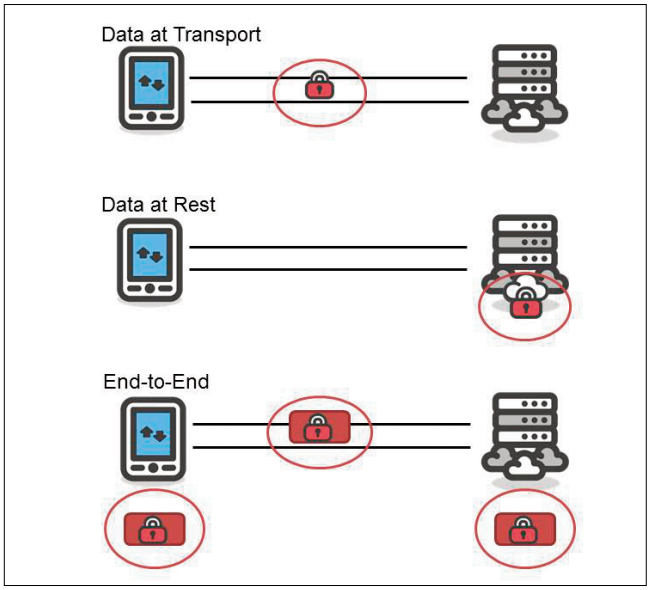
Es gilt heute ausnahmslos: Werden personenbezogene Daten über das Internet übertragen, müssen diese immer verschlüsselt werden. Die Art der Verschlüsselung hängt dann allerdings von der Sensibilität der Daten ab. Von der Methodik unterscheidet man zwischen Data-at-Transport (Transportverschlüsselung), Data-at-Rest (Ruheverschlüsselung) und End-to-End (Ende-zu-Ende-Verschlüsselung) (Bild 3).

Data-at-Transport (Transportverschlüsselung)

Die Übermittlung zwischen der App und dem Backend-Server beziehungsweise Cloud-Dienst wird durch eine SSL/TLS-Verschlüsselung abgesichert. Wird dabei das HTTP-Protokoll verwendet, dann spricht man von HTTPS. Da dieses Protokoll sehr komplex und (in der Denkweise des Internets) sehr alt ist, kann man beim Betrieb viel falsch machen.

Der Vorteil bei der Anwendung mit einer App ist, dass die Restriktionen des Browsers, nämlich möglichst viele Nutzer auch mit veralteten Softwareständen zu erreichen, nicht gelten. Neben dem Einsatz von aktuellen Protokollversionen (momentan TLS1.2) wird der Begriff Perfect Forward Secrecy (PFS) häufig diskutiert. Darunter versteht man eine Konfiguration der Transportverschlüsselung in der Weise, dass nur bestimmte kryptografische Algorithmen (Cipher-Suites) zum Einsatz kommen.

Diese haben die Eigenschaft, dass die für eine verschlüsselte Kommunikation notwendigen gemeinsamen Verschlüsselungsschlüssel mit einem speziellen Verfahren (dem sogenannten Diffie-Hellmann-Schlüsseltausch) ausgetauscht werden. Das Besondere daran ist, dass ein Angreifer, der sich als Man in the Middle in der Verbindung befindet, trotzdem nicht auf den Verschlüsselungsschlüssel kommen kann.



Strukturierung von Verfahren zur Verschlüsselung (Bild 3)

Eine weitere Eigenschaft, die gerade im Zeitalter der globalen Überwachung durch Geheimdienste bedeutend ist, führt dazu, dass selbst kein Knacken einer verschlüsselten Verbindung oder bei Entwenden des privaten SSL-Server-Schlüssels (mit der Heartbleed-Lücke 2014 war dies möglich), keine massenhafte Entschlüsselung von auf Vorrat gespeicherten Kommunikationsinhalten möglich ist.

Der Einsatz von Perfect Forward Secrecy wird von manchen (deutschen) Datenschutzaufsichtsbehörden als Stand der Technik angesehen und muss deswegen umgesetzt werden – der Aufwand besteht im Allgemeinen in einer Konfiguration des Webservers beziehungsweise des Load-Balancers/der Firewall und kostet damit fast nichts. Eine kleine Checkliste für den datenschutzkonformen Einsatz von SSL/TLS mit noch einigen weiteren Prüfpunkten ist in Tabelle 1 dargestellt.

Im Web stehen zahlreiche Dienste zur Überprüfung der Passwortqualität zur Verfügung (Bild 4)

Data-at-Rest (Ruheverschlüsselung)

Wurden personenbezogene Daten von einer App an das Backend übertragen, müssen diese weiterhin verschlüsselt vorgehalten werden. Zumindest dann, wenn diese nicht aktiv für eine Verarbeitung benötigt werden. In diesem Fall spricht man von einer Ruheverschlüsselung. Konkret versteht man darunter eine Vollverschlüsselung des eingesetzten Filesystems oder die Aktivierung der Verschlüsselung der Datenbank.

Tabelle 1: Checkliste für die Transportverschlüsselung

Best-Practice-Checkliste SSL/TLS bei Einsatz von mobilen Anwendungen (Apps)	
Ausschließlich Protokollversion TLS1.2 (kein SSL3, TLS1.0, TLS1.1)	o
Ausschließlich Cipher-Suites, die Perfect Forward Secrecy (PFS) unterstützen	o
SSL-Pinning (empfohlen ist Public-Key-Pinning)	o
SSL-Zertifikat von einem vertrauenswürdigen Anbieter	o
Zertifikate mit 4096-Bit-RSA	o
Kein SHA-1 bei der Signierung von Zertifikaten (besser SHA-256)	o
Keine veralteten kryptografischen Algorithmen (zum Beispiel RC4, DES ...)	o
Keine Überstützung von unverschlüsseltem HTTP auf demselben Server (Gefahr von sogenannten SSL-Stripping-Angriffen)	o
Aktuelles und systematisches Patch-Management (zum Beispiel wegen Lücken wie Heartbleed oder ShellShock)	o
Verwendung von TLS-Client-Zertifikaten zwischen App und Backend	o

Zu beachten ist bei dieser Form der Verschlüsselung, dass der Betreiber des Backends (zum Beispiel der Cloud-Service) weiterhin auf die Daten zugreifen könnte, wenn dieser es darauf anlegen würde. Unter der Brille des Datenschutzes führt dies dazu, dass in diesem Fall rechtliche Regelungen zur Übermittlung (in ein unsicheres Drittland, sofern dies ein US-Anbieter ist) getroffen werden müssen.

End-to-End (Ende-zu-Ende-Verschlüsselung)

Diese Königklasse der Verschlüsselung führt dazu, dass die Daten in der App verschlüsselt und erst beim Empfänger (und nicht zum Beispiel im Webserver, sondern beispielsweise im Applikationsserver) wieder entschlüsselt werden. Verwendet werden hierfür asymmetrische Verschlüsselungsverfahren wie RSA oder – momentan auch zunehmend eingesetzt – sogenannte elliptische Kurven.

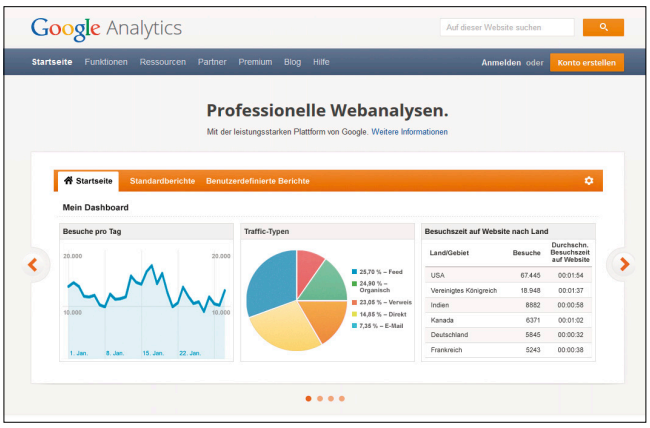
Die Implementierung dieser Algorithmen sollte tunlichst nicht selbst erfolgen, da durch einfache Implementierungsfehler ein Knacken des Verfahrens zumindest für Experten relativ leicht zu bewerkstelligen ist. Knackpunkt für den praktikablen Einsatz ist die Verwaltung der kryptografischen Schlüssel, die beim App-Betreiber eine Public-Key-Infrastruktur voraussetzt.

Da damit gewisse Kosten und Aufwände verbunden sind, sehen momentan die Aufsichtsbehörden diese Verschlüsselungstechnik eher bei sehr sensiblen Daten wie Gesundheitsdaten als erforderlich an – die Übermittlung von Login-Daten bei einer Spiele-App wäre statt dessen ausreichend mit einer Transportverschlüsselung gesichert.

Authentifizierung

Viele mobile Anwendungen benötigen ein Nutzerkonto und damit eine Anmeldung des Nutzers. Während die Identifikation bei der Anmeldung meist mit einer E-Mail-Adresse (oder mit Hilfe eines bei dem ersten Start der App generierten Tokens) erfolgt, muss diese mit einem Authentifizierungsverfahren nachgewiesen werden.

Heute am weitesten verbreitet ist nach wie vor das Passwort. Bei dessen Einsatz gilt es zu beachten, dass dieses bei Verwendung einer Passwort-Merken-Funktion nicht im Klartext innerhalb des App-lokalen Speicherbereichs des Dateisystems abgelegt werden sollte. Auch auf dem Backend- ▶



Google Analytics ist eines der am häufigsten eingesetzten Tracking-Verfahren (Bild 5)

System ist es Stand der Technik, dass Passwörter nicht im Klartext, sondern als Salted-Hash-Werte in der Datenbank abgelegt werden. Ein eingegebenes Passwort wird dann ebenfalls in diesen Hashwert transformiert – sind beide Hashwerte identisch, war das Passwort korrekt (Bild 4).

Dieses Verfahren hat den Vorteil, dass weder der App-Betreiber noch ein Angreifer, der die Nutzerdatenbank hackt, Zugriff auf das Klartextpasswort bekommt – vorausgesetzt, das Hashverfahren ist für eine Passwortspeicherung geeignet. Heute werden häufig noch die Hashverfahren MD5 oder SHA eingesetzt, um einen Hashwert zu generieren.

Damit dieser durch eine Brute-Force oder Wörterbuch-Attacke nicht mit viel Rechenaufwand (den aktuelle Grafikkarten bestens leisten) zurückgerechnet werden kann, muss das Passwort möglichst lang und komplex sein.

Standard-Hashverfahren

Bei Verwendung von Standard-Hashverfahren geht man heute davon aus, dass diese mindestens 10-stellig und Groß-/Kleinbuchstaben, Ziffern und Sonderzeichen enthalten müssen. Aus Sicht der Usability und der Merkbarkeit der Passwörter ist dies nicht optimal.

Eine leichte Verbesserung kann durch den Einsatz von Hashverfahren erreicht werden, die relativ langsam berechnet werden können – ein Backend-Server wird kaum Unterschiede merken, ein Angreifer allerdings schon. Empfohlen sei hier, sich mit Verfahren wie bcrypt oder PBKDF-2 zu beschäftigen.

Aus Datenschutzsicht gibt es noch ein weiteres Entgegenkommen an die Bedienbarkeit. Der Grundsatz des informationellen Selbstbestimmungsrechts besagt auch, dass jeder Nutzer über die Verwendung seiner personenbezogenen Daten und (in sehr engen Grenzen) über sein Bedürfnis nach Sicherheit selbst entscheiden kann: Wird beispielsweise durch eine Passwort-Gütefunktion die Sicherheit des Passwortes visualisiert (zum Beispiel rot, gelb, grün), dann kann jeder Nutzer selbst die Entscheidung treffen, ob dieser lieber die Bedienbarkeit oder den Schutz gegen Angreifer im Vordergrund sehen möchte.

Wird eine App entwickelt, mit der sehr sensible personenbezogene Daten verarbeitet werden (zum Beispiel Gesundheitsdaten), dann ist im Allgemeinen eine Ein-Faktor-Authentifizierung ausschließlich mittels Passwort zur Sicherstellung eines dem Schutzbedarfs angemessenen Sicherheitsniveaus nicht ausreichend.

Wirksame Verfahren wie der Einsatz eine Kryptochips in einer Speicherkarte (zum Beispiel SIM) sind noch nicht weit genug verbreitet, um eine praktikable Anwendbarkeit sicherzustellen.

Ein One-time-Passwort per SMS ist zu unsicher, da diese teilweise trivial von einer auf dem mobilen Endgerät installierten Malware-Anwendung abgefangen werden können. Sollte eine solche Anwendung entwickelt werden, dann empfiehlt es sich, die gesetzlich festgeschriebenen Beratungsleistungen der Datenschutzaufsichtsbehörde in Anspruch zu nehmen, ehe immense Kosten oder der erzwungene Stopp einer Anwendung auf einen App-Betreiber zukommen.

Lokale Speicherung von Daten

Die meisten Anwendungen speichern benötigte Daten im App-lokalen Speicherbereich der mobilen Anwendung. Gegen diesen Ansatz ist auch im Allgemeinen nicht zu sagen. Allerdings ist bei der Entwicklung darauf zu achten, dass diese geschützten Speicherbereiche auch verwendet werden.

So ermöglichen beispielsweise Android-Apps die Verwendung des Speicherplatzes auf der SD-Karte – eine App, die zum Beispiel abfotografierte Arzt-Rezepte dort unverschlüsselt ablegt und diese Dateien bei Deinstallation nicht löscht, löst datenschutzrechtliche Verstöße samt Einschaltung der Aufsichtsbehörde aus.

Auch die Frage, inwiefern App-lokale Informationen Bestandteile der Cloud-basierten Backup-Strategie des Endge-

Information des BayLDA: Orientierungshilfe für App-Entwickler (Bild 6)

räts werden, muss beachtet werden. Hier bietet es sich an, die lokalen Informationen nur verschlüsselt zu speichern und den Encryption-Key explizit aus dem Cloud-Backup auszunehmen. Ein weiterer Vorteil ist dabei, dass im Fall solcherart verschlüsselter Backups in der Cloud keine datenschutzrechtliche Übermittlung in ein (aus Datenschutzsicht) unsicheres Drittland vorliegt, sofern ein US-Cloud-Anbieter verwendet wird.

Tracking

Die Art und Weise, wie eine App verwendet wird, ist für die Produktverbesserung und Fehlerbeseitigung sehr wichtig. Dazu werden häufig sogenannte Tracking-Verfahren wie Google Analytics oder Crashlytics eingesetzt (Bild 5).

Auch Anwendungen, die kostenlos aus dem App-Store geladen werden und auf einer Finanzierung durch Werbung aufbauen, benötigen meist irgendeine Form einer sogenannten Nutzungsprofilbildung des Anwenders. Hierbei ist zu beachten, dass der Datenschutz immer tangiert ist – auch wenn nur Diagnosedaten wie Stacktrace, Geräteinformationen oder Softwarestände übertragen werden (Stichwort IP-Adresse).

Dies bedeutet aber nicht, dass solche Arten von Datenerhebungen per se unzulässig sind. Stattdessen muss unterschieden werden, welche personenbezogenen Daten übermittelt werden:

Sind Daten wie Kundennummern, E-Mail-Adressen, eindeutige Geräte-IDs (zum Beispiel Hash einer IMEI-Nummer) Bestandteil der Datenübermittlung, dann muss zum Beispiel die Einwilligung des Nutzers vor der Übermittlung geholt werden (Opt-in).

Sind nur Nutzungsdaten der App (Startzeitpunkt, Navigation innerhalb der App, ausschließlich technische Fehlerprotokolle) Bestandteil der Übermittlung und wird die Eindeutigkeit der Installation durch eine zufällige Kennung (erzeugt bei erstem Start der App) dargestellt, dann spricht man von einem pseudonymen Nutzungsprofil. Diese dürfen zum Zweck der Produktverbesserung oder Werbung dann eingesetzt werden, wenn der Nutzer darüber informiert wird und für diesen eine Möglichkeit zum Wiederspruch existiert (Opt-out).

Bei allen Verfahren muss sichergestellt werden, dass die IP-Adresse nur für den Transport über das Internet verwendet wird – schon eine Geolokalisierung wird als Verarbeitung personenbezogener Daten angesehen und bedarf einer Rechtsgrundlage. Tipp: Die IP-Adresse vor der serverseitigen Verarbeitung durch Löschen des letzten Oktetts (bei IPv4) anonymisieren – dann steht einer genauso guten Geolokalisierung nichts mehr im Weg.

Datenschutzaufsichtsbehörden

Werden die dargestellten Anforderungen beachtet, dann wird das Risiko deutlich reduziert, in ein aufsichtliches Verfahren mit der Aufsichtsbehörde verwickelt zu werden.

Die behördlichen Möglichkeiten sind allerdings (bald) nicht zu vernachlässigen: Technische Verstöße sind zwar momentan meist nicht bußgeldbewährt, können aber per Zwangsmaßnahme (und Zwangsgelder in spürbarer Höhe) angeordnet werden. Rechtliche Verstöße wie das Fehlen einer Datenschutzerklärung oder eine ungeregelte Übermittlung in unsichere Drittstaaten können heute schon mit theoretisch bis zu 50.000/300.000 Euro sanktioniert werden.

Zukünftig wird sich in diesem Bereich allerdings viel ändern. So wird ab Juni 2018 die Europäische Datenschutzgrundverordnung einzuhalten sein, die erstmalig auch Verstöße im technischen Bereich mit bis zu 10 Millionen Euro oder 2 Prozent des weltweiten Umsatzes sanktionsfähig macht.

Links zum Thema

- Orientierungshilfe für App-Entwickler
  https://www.lda.bayern.de/oh-apps.pdf
- Prüfkatalog Apps
  https://www.lda.bayern.de/pruefliste-apps.pdf

Orientierungshilfe für App-Entwickler

Weitere Informationen zur momentanen deutschen Gesetzeslage sowie zu technischen Grundanforderungen finden sich (auch in Englisch) unter www.lda.bayern.de (Bild 6). Ein Prüfkatalog der bayerischen Datenschutzaufsichtsbehörde für den nichtöffentlichen Bereich (BayLDA), der von diesen bei der Bewertung von bayerischen Apps die Basis bildet, kann ebenfalls dort heruntergeladen werden.

Ansonsten ist zu empfehlen, sich bei Unsicherheiten oder Fragen rechtzeitig an die zuständige Datenschutzaufsichtsbehörde zu wenden – dies hilft, Kosten zu sparen, das Projektrisiko zu minimieren, und, auch nicht zu vernachlässigen, eine App zu entwickeln, die konform zu den deutschen Datenschutzgesetzen ist.

Fazit

Datenschutz ist eine Anforderung, die bei der Entwicklung einer mobilen Anwendung rechtzeitig beachtet werden muss. Davon betroffen sind sowohl rechtliche als auch technische Aspekte, die häufig Hand-in-Hand gehen. Deswegen ist es essenziell, dieses Themenfeld nicht ausschließlich den Rechtsabteilungen zu überlassen sondern – am besten gemeinsam mit dem Datenschutzbeauftragten – auch aus Entwicklersicht auf eine datenschutzkonforme Ausgestaltung zu achten. ■

Andreas Sachs
ist Informatiker und leitet das Referat IT-Sicherheit und technischer Datenschutz beim Bayerischen Landesamt für Datenschutzaufsicht in Ansbach.
www.lda.bayern.de

E-MAIL-MARKETING

Spam oder Werbe-Mail

Die Grenze zwischen Spam- und rechtmäßiger Werbe-Mail ist nicht leicht zu ziehen.

Verschickt Unternehmer U. eine Werbe-Mail an Person P., ohne dass dem Erhalt dieser E-Mail vorab ausdrücklich zugestimmt wurde, so stellt dies ein unlauteres Verhalten dar. Als solches ist es regelmäßig zugleich auch ein kostenpflichtig abmahnbarer Verstoß gegen das Wettbewerbsrecht. Dabei spielt es keine Rolle, ob P. Privatperson oder ebenfalls Unternehmer ist. Im erstgenannten Fall verstieße die Spam gegen das allgemeine Persönlichkeitsrecht, im letztgenannten wäre es ein unzulässiger Eingriff in den eingerichteten und ausgeübten Gewerbebetrieb. In jedem Fall ist bereits der Versand einer einzigen Spam-Mail rechtswidrig. Dies gilt nicht nur für klassische Werbung, sondern beispielsweise auch für Kundenzufriedenheitsanfragen, so hat es jedenfalls das Oberlandesgericht (OLG) Dresden mit Urteil vom 24. April 2016 (Aktenzeichen: 14 U 1773/13) entschieden.

Der Streitwert für gerichtliche Auseinandersetzungen aufgrund der Spam-Problematik beträgt zwar in aller Regel weniger als 5000 Euro, oftmals sogar unter 3000 Euro (vgl. OLG Frankfurt am Main, Beschluss vom 2. März 2016, Aktenzeichen: 6 W 9/16), löst aber dennoch nicht unerhebliche Kosten aus. Zudem besteht immer die Gefahr, insbesondere beim Versand eines Newsletters an Hunderte oder gar Tausende Kunden, dass es nicht bei einem einzigen Rechtsstreit bleibt.

Werbung oder nicht?

Der gesetzliche Rahmen für den Versand elektronischer Werbung besteht insbesondere aus folgenden Regelwerken:

- Gesetz gegen den unlauteren Wettbewerb (UWG),
- Bundesdatenschutzgesetz (BDSG),
- Telemediengesetz (TMG),
- Gesetz über elektronische Handelsregister und Genossenschaftsregister sowie das Unternehmensregister (EHUG).

Im Kern bedeutet dies also, dass E-Mails mit werblichem Inhalt bestimmte Mindestanforderungen erfüllen müssen. Insbesondere gibt es diverse Pflichtinhalte sowie das Erfordernis, die Einwilligung der potenziellen Empfänger einzuholen und im Idealfall auch gerichtsfest zu dokumentieren.

Mail-Inhalte sind dann als Werbung einzustufen, wenn sie im Zuge der Ausübung eines Handels, Gewerbes, Handwerks oder freien Berufs mit dem Ziel getätigt werden, den Absatz zu fördern. Bei Licht betrachtet handelt es sich also um direkte oder auch mittelbare Werbung für das eigene Unternehmen, für die eigenen Produkte und/oder für die eigenen Dienstleistungen. Es kommt dabei nicht darauf an, ob die betreffenden Produkte beziehungsweise Dienstleistungen entgeltlich oder kostenfrei angeboten werden. Ausnahmen bilden lediglich reine System- beziehungsweise Statusmeldungen, wie etwa über den Eingang der Bestellung, über den Erhalt der Zahlung oder über den Versand der Ware.

Double-opt-in

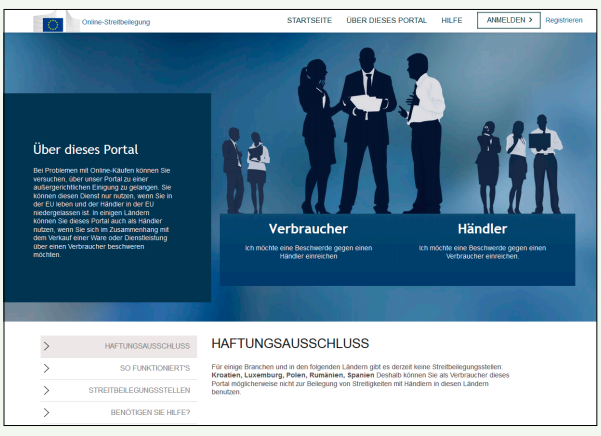
Rechtmäßige Werbe-Mails bedürfen stets – mit wenigen Ausnahmen – der vorherigen und ausdrücklichen Zustimmung des Empfängers. Dieser muss selbst aktiv werden. Die Einwilligungserklärung darf nicht durch den Versender eingeholt werden. In aller Regel heißt es für die werbenden Unternehmen also abwarten.

Um juristisch auf der sicheren Seite zu sein, sollten die Vorgaben des sogenannten Double-opt-in-Verfahrens eingehalten werden. Dies bedingt die folgende Vorgehensweise:

1. Der potenzielle Empfänger muss die Werbung, den Newsletter et cetera selbst aktiv anfordern. Dies kann er zum Beispiel durch Eintragung seiner Mail-Adresse in das entspre-

Praxis-Tipp

Betreiber von Websites, die zwar keinen klassischen Online-Shop anbieten, allerdings per Mail-Kommunikation auf Kundenwünsche eingehen beziehungsweise auf diesem Weg Verträge abschließen, müssen den Hinweis auf die OS-Plattform in ihre Mail-Signatur einbinden. Dieser Hinweis kann etwa wie folgt formuliert werden: »Infos zur Online-Streitbeilegung: Die Internetplattform zur Online-Beilegung von Streitigkeiten der EU (OS-Plattform) ist unter folgendem Link erreichbar: www.ec.europa.eu/consumers/odr.«



Ein Hinweis auf die OS-Plattform sollte in der E-Mail-Signatur nicht fehlen

chende Anforderungsformular für einen Newsletter tun.

2. Anschließend ist eine erste E-Mail an die angegebene Adresse zu versenden, die einen Aktivierungs-Link enthält. Durch Anklicken dieses Links wird die eingetragene Adresse durch den tatsächlichen Inhaber verifiziert.
3. Erst und nur nach erfolgreicher Verifizierung darf mit dem Versand begonnen werden.
4. Jede einzelne Werbe-Mail muss unter anderem auch einen ausdrücklichen Hinweis darauf enthalten, dass und wie eine Beendigung des Mail-Bezugs möglich ist.

Alle vier genannten Schritte sind in dieser Abfolge zu realisieren. Eine rechtswirksame Einwilligung setzt detaillierte Informationen voraus, nämlich im Einzelnen über die Natur der Inhalte, die Häufigkeit des Mail-Versands, die Einhaltung von Datenschutzvorschriften und eventuell beteiligte Partner-Unternehmen. Diese Informationen sollten daher unterhalb des Eingabefelds für die Mail-Adresse gut sichtbar platziert werden. Ein Verweis, zum Beispiel in Form eines sprechenden Links, auf die eigene Datenschutzerklärung sollte ebenfalls nicht fehlen.

Ausnahmeregelung

Im UWG ist ein Ausnahmefall des Double-opt-in-Verfahrens geregelt. Unter bestimmten Voraussetzungen kann der Versand von Mail-Werbung auch ohne ausdrückliche Einwilligung des Empfängers erfolgen:

- E-Mail-Adresse im Zusammenhang mit dem Verkauf von Waren oder Dienstleistungen erhalten,
- Direktwerbung für eigene ähnliche Waren,
- kein Widerspruch durch Empfänger,
- klarer Hinweis auf jederzeitige Widerspruchsmöglichkeit.

Letztlich stellt sich hierbei die Frage, wann eine Ähnlichkeit von Waren beziehungsweise Dienstleistungen vorliegt. Davon ist dann auszugehen, wenn die betreffenden Waren beziehungsweise Dienstleistungen generell austauschbar sind beziehungsweise gleichen oder ähnlichen Zwecken dienen. Für die Bewertung dieser Punkte darf jedoch kein allzu niedriger Maßstab angelegt werden, da es sich hierbei ausdrücklich um eine Ausnahmeregelung handelt.

Egal ob mit Double-opt-in-Verfahren oder unter Nutzung der genannten Ausnahmeregelung – der Versender von elektronischer Werbung muss im Zweifelsfall die jeweiligen Voraussetzungen belegen können. Daher sollten die Anmeldung beziehungsweise Eintragung, der Versand der Bestätigungs-Mail, die Verifizierung der Mail-Adresse oder alternativ die Voraussetzungen für das Vorliegen der Ausnahmeregelung akribisch protokolliert werden.

Inhaltlich gibt es ebenfalls einiges zu beachten. Jede einzelne Werbe-Mail sollte die folgenden Eckpfeiler aufweisen:

- Angaben zum Absender,
- eindeutiger Hinweis auf Werbung schon im Betreff,
- Werbe-Inhalt und erteilte Einwilligung müssen inhaltlich übereinstimmen,
- keine wettbewerbswidrigen Produktangaben,
- korrekte Preisangaben,
- Hinweis, dass und wie ein Widerruf der Werbe-Einwilligung möglich ist.

Übrigens: Eine erteilte Einwilligung ist nicht unendlich lange wirksam, sie gilt vielmehr nur für etwa ein bis zwei Jahre.

Links zum Thema

- Video-Trainings des Autors
  www.video2brain.com/de/trainer/michael-rohrlich
- Blog des Autors zum Thema Online-Recht für Webmaster
  http://webmaster-onlinerecht.de
- Blog des Autors zum Verbraucherrecht online
  http://verbraucherrechte-online.de
- Weitergehende Informationen zum Thema E-Commerce
  http://rechtssicher.info

Mail-Unterschrift

Im Hinblick auf die Angaben zum Absender sind die Vorschriften des TMG und des EHUG zu befolgen. Immer dann, wenn die Werbe-Mails zumindest auch an Bestandskunden gerichtet sind, müssen vergleichsweise umfangreiche Informationen angegeben werden. Eine Mail-Signatur, also sozusagen die abschließende Mail-Unterschrift, mit dem Mindest-Inhalt gemäß EHUG könnte also wie folgt aussehen:

```
Mustermann GmbH
Sitz der Gesellschaft: Musterhausen
Registergericht: AG Musterhausen, HRB 12345
Geschäftsführer: Max Mustermann
```

Zusätzlich können beziehungsweise müssen gemäß TMG bisweilen noch weitere Angaben gemacht werden. Werden nicht nur ausnahmsweise Verträge via E-Mail abgeschlossen, dann ist zusätzlich noch der Hinweis auf die Online-Plattform zur alternativen Streitbeilegung (so genannte OS-Plattform) der Europäischen Kommission hinzuzufügen.

Der Versand von Spam-Mails wird in der überwiegenden Zahl der vorkommenden Fälle als Verstoß gegen das Wettbewerbsrecht eingestuft. Als solcher kann er unterschiedliche Konsequenzen haben, zum einen kostenpflichtige Abmahnungen von konkurrierenden Unternehmen oder Verbraucherschutzorganisationen, und zum anderen Geldbußen. ■

Michael Rohrlich
ist Rechtsanwalt und Fachautor aus Würselen. Seine beruflichen Schwerpunkte liegen auf dem Gebiet des Online-Rechts und des gewerblichen Rechtsschutzes.
www.rechtssicher.info

ARBEITSMARKT

TRENDS UND JOBS FÜR ENTWICKLER

Monatliches Ranking

Leichter Rückgang im Sommer

Im Sommer geht die Nachfrage nach Entwicklern erfahrungsgemäß leicht zurück. Das ist auch diesmal so. Betrachtet man die Verteilung der Jobangebote, so ist Bayern mit rund 20 Prozent das Bundesland, für das die meisten Entwickler gesucht werden – sowohl Webentwickler als auch Entwickler von Apps für Mobilgeräte. Auf den Plätzen zwei und drei landen NRW und Baden Württemberg.

Jobs für Web- und Mobile-Entwickler

Auf diese Top-3-Länder entfallen 56 Prozent der Angebote für Webentwickler und knapp 50 Prozent der Gesuche nach Mobile-Entwicklern (Bild 1).

Betrachtet man die deutschen Großstädte, so liegen München, Berlin und Hamburg wie üblich ganz vorne, gefolgt von Frankfurt, Köln und Düsseldorf. Bei den Mobile-Entwicklern liegen derzeit Hannover und Stuttgart noch vor Düsseldorf.

Programmiersprachen

In der Abfrage der Programmiersprachen bei Jobkralle.de (Bild 2) steckt eine Besonderheit und in diesem Monat zusätzlich noch eine inhaltliche Änderung.

Die Besonderheit: Im Gegensatz zu den anderen Abfragen werden hier nur Angebote berücksichtigt, welche den Namen der Sprache im Titel der Stellenausschreibung tragen, beispielsweise *Anwendungsentwickler C#*. Wird die Programmiersprache lediglich irgendwo im Text erwähnt, zählt das betreffende Angebot nicht als Treffer.

Die inhaltliche Änderung besteht darin, dass diesmal die Entwicklung von iOS-Programmen über den Begriff *iOS* abgefragt wurde. Das ist formal nicht korrekt – schließlich kann man für iOS mit mehreren Sprachen entwickeln – aber ganz offensichtlich werden Swift-Entwickler oder Entwickler, die noch Objective-C nutzen, nicht über die Namen der Programmiersprachen gesucht. Die Abfrage von Swift lieferte lediglich 13 Treffer (bereinigt um die Treffer für das Finanztransaktionssystem gleichen Namens), und für Objective-C lieferte Jobkralle.de überhaupt keine Treffer.

Die Suche nach dem Begriff *iOS* unter gleichen Bedingungen zeigte aber 424 Treffer, beispielsweise als *iOS Engineer* oder als *Software Engineer iOS Mobile Apps*.

Technologien

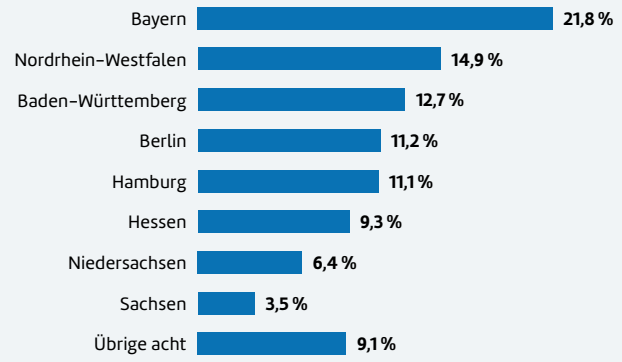
Die Ergebnisse der aktuellen Abfrage der in den Stellenangeboten angesprochenen Technologien haben sich im Vergleich zum Vormonat nur sehr wenig verändert (Tabelle 1). Ganz oben liegen Cloud, MySQL und HTML5, gefolgt von SharePoint.

Tabelle 1: Technologie

Rang	Technologie	Anteil *
1	Cloud	17,8 %
2	MySQL	12,6 %
3	HTML5	10,7 %
4	SharePoint	7,7 %
5	Big Data	7,1 %
6	Android	7,0 %
7	Microsoft SQL Server	6,7 %
8	iOS	6,0 %
9	CSS3	5,3 %
10	AngularJS	4,2 %
11	Windows 10	3,4 %
12	WPF	3,1 %
13	NoSQL	2,6 %
14	Responsive Web	2,5 %
15	WCF	1,7 %
16	Azure	1,6 %

* Prozentualer Anteil der Treffer

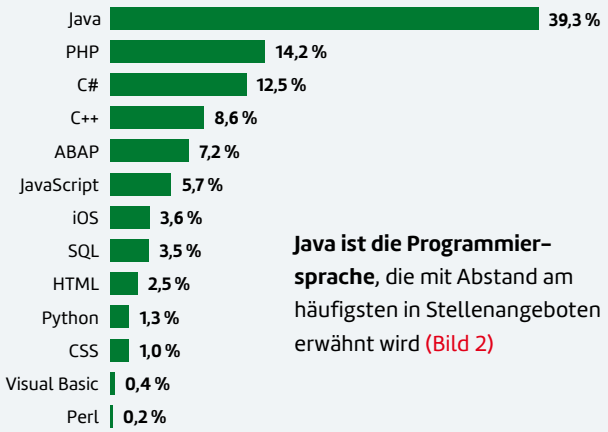
Jobs für Mobile-Entwickler



Auf die Top 3 der Bundesländer entfallen knapp die Hälfte der Jobangebote für Mobile-Entwickler (Bild 1)

web & mobile developer 8/2016 Quelle: Eigene Erhebungen, Jobkralle.de

Programmiersprachen



Java ist die Programmiersprache, die mit Abstand am häufigsten in Stellenangeboten erwähnt wird (Bild 2)

web & mobile developer 8/2016 Quelle: Eigene Erhebungen, Jobkralle.de

Zahl des Monats

Außerhalb ihrer Arbeitszeit programmieren **88,6 Prozent** der Entwickler mindestens eine Stunde pro Woche an Open-Source- oder Hobbyprojekten. **19,5 Prozent** investieren fünf bis zehn Stunden pro Woche in ihrer Freizeit in diese Projekte, **5,6 Prozent** sogar über 20 Stunden.

Quelle: Stack Overflow Entwicklerumfrage 2016

Solcom

Freiberufler arbeiten länger

Der Technologiedienstleister Solcom hat zwischen März und Mai 2016 die Abonnenten seines Magazins befragt. Aus den Antworten wurde die Marktstudie »Freiberufler: Arbeit und Leben im Einklang?« generiert. Ein Großteil der befragten Freiberufler arbeitet mehr als 40 Stunden pro Woche, jeder Vierte sogar mehr als 50 (Bild 3). Trotzdem empfindet über die Hälfte ihre Tätigkeit als gut mit dem Familien- beziehungsweise Privatleben vereinbar, nur jeder Fünfte macht dabei andere Erfahrungen. Entsprechend sieht auch die Mehrheit der Umfrageteilnehmer ihre Selbstständigkeit als besser vereinbar mit dem Familien- beziehungsweise Privatleben als eine Festanstellung. Ein Drittel sieht es jedoch genau umgekehrt.

Für vier von zehn Befragten waren die besseren Möglichkeiten bei der Vereinbarkeit von Beruf und Privatleben ein Grund, in die Freiberuflichkeit zu wechseln, nur jeder Zehnte würde den umgekehrten Weg zurück in die Festanstellung gehen. Die größten Schwierigkeiten bei der Vereinbarkeit werden dabei in der Reisetätigkeit gesehen, gefolgt von der Arbeitszeitbelastung im Projekt und der wirtschaftlichen Unsicherheit. Angesichts dieser Aussagen ist es nicht verwunderlich, dass drei Viertel der befragten Freiberufler Selbstbestimmung als wichtigsten Wert angeben, nach dem sie ihr Leben ausrichten (Bild 4), gefolgt von Erfolg und Anerkennung.

www.solcom.de

StepStone

Fachkräfte kennen ihren Marktwert

Bei der Jobsuche achten Fachkräfte ganz besonders auf das Image der Unternehmen, bei denen sie anheuern wollen. Acht von zehn Fachkräften würden sogar eine passende Stelle ausschlagen, wenn sie sich nicht mit dem neuen Unternehmen identifizieren können, das die Stelle ausgeschrieben hat.

Das meldet die Jobbörse StepStone.de als eines der Ergebnisse einer Studie mit mehr als 14.000 Fach- und Führungskräften.

Das große Selbstbewusstsein der Fachkräfte leitet sich unter anderem aus ihrer aktuell starken Position auf dem Arbeitsmarkt ab. Mehr als zwei Drittel der Befragten betrachten ihren aktuellen Arbeitsplatz als gesichert. Sieben von zehn sind sich sicher, bei Bedarf in maximal einem halben Jahr eine neue Stelle finden zu können.

Ein Drittel der Fachkräfte geht sogar davon aus, die Jobsuche innerhalb von nur drei Monaten erfolgreich abzuschließen.

www.stepstone.de

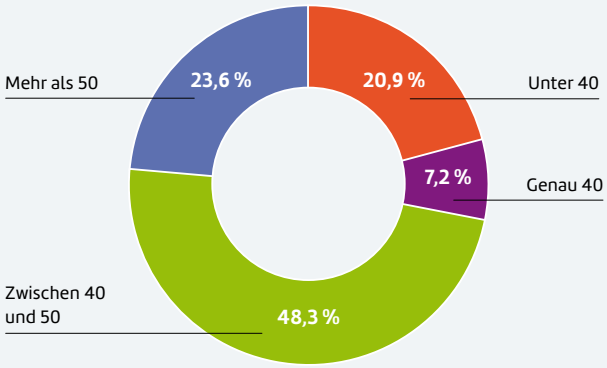
Springer-Fachbuch

Erfolgreiches IT-Recruiting

Autor Frank Rechsteiner liefert in diesem Fachbuch neue Strategien und Handlungsempfehlungen zur Rekrutierung von IT-Spezialisten. Basierend auf zahlreichen Umfragen und Studien zeigt er Trends für den Arbeits- und Bewerbermarkt im IT-Bereich auf und gibt Tipps zum erfolgreichen Umgang mit den derzeitigen Herausforderungen. Der Autor schlägt HR-Verantwortlichen vor, spezialisierte IT-Mitarbeiter nicht nur als Ressource zu betrachten, sondern auf die besonderen Bedürfnisse dieser Berufsgruppe einzugehen. Die daraus abgeleiteten Maßnahmen ermöglichen es, geeignete Experten zu finden und dauerhaft an das Unternehmen zu binden. Das Buch ist als Softcover-Ausgabe unter der ISBN 978-3-658-13157-9 zu finden.

www.springer.com/de

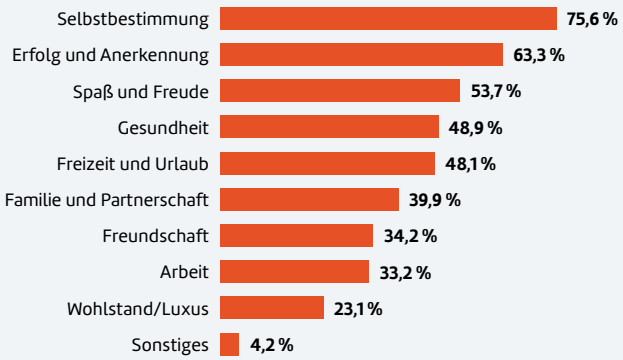
Freiberufler: Stunden pro Woche



Weniger als ein Drittel der Freiberufler kommt mit einer 40-Stunden-Woche aus (Bild 3)

web & mobile developer 8/2016 Quelle: www.solcom.de

Freiberufler: Die wichtigsten Werte

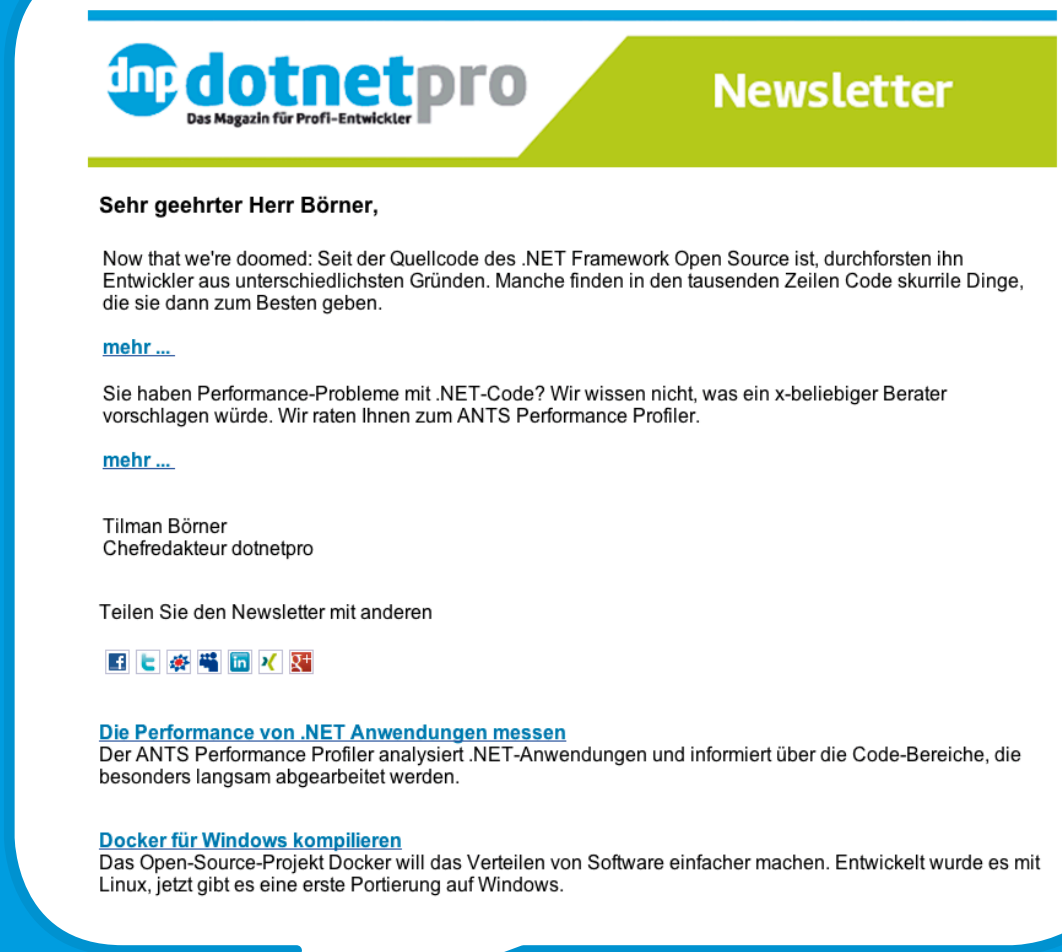


Selbstbestimmung sowie Erfolg und Anerkennung sind für Freiberufler am wichtigsten (Bild 4)

web & mobile developer 8/2016 Quelle: www.solcom.de

dotnetpro Newsletter

Top-Informationen für den .NET-Entwickler.
Klicken. Lesen. Mitreden.


dotnetpro
Das Magazin für Profi-Entwickler
Newsletter

Sehr geehrter Herr Börner,

Now that we're doomed: Seit der Quellcode des .NET Framework Open Source ist, durchforsten ihn Entwickler aus unterschiedlichsten Gründen. Manche finden in den tausenden Zeilen Code skurrile Dinge, die sie dann zum Besten geben.

mehr ...

Sie haben Performance-Probleme mit .NET-Code? Wir wissen nicht, was ein x-beliebiger Berater vorschlagen würde. Wir raten Ihnen zum ANTS Performance Profiler.

mehr ...

Tilman Börner
Chefredakteur dotnetpro

Teilen Sie den Newsletter mit anderen

Die Performance von .NET Anwendungen messen
Der ANTS Performance Profiler analysiert .NET-Anwendungen und informiert über die Code-Bereiche, die besonders langsam abgearbeitet werden.

Docker für Windows kompilieren
Das Open-Source-Projekt Docker will das Verteilen von Software einfacher machen. Entwickelt wurde es mit Linux, jetzt gibt es eine erste Portierung auf Windows.

Jetzt kostenlos anmelden:

dotnetpro.de

twitter.com/dotnetpro_mag

facebook.de/dotnetpro

gplus.to/dotnetpro

Anbieterverzeichnis

für Deutschland, Schweiz und Österreich.

Consulting / Dienstleister

ANEXIA Internetdienst-leistungs GmbH
Feldkirchner Straße 140
9020 Klagenfurt / AUSTRIA
T +43-50-556
F +43-50-556-500
info@anexia-it.com

ANEXIA wurde im Juni 2006 von Alexander Windbichler als klassischer Internet Service Provider gegründet. In den letzten Jahren hat sich ANEXIA zu einem stabilen, erfolgreichen und international tätigen Unternehmen entwickelt, das namhafte Kunden rund um den Globus mit Standorten in Wien, Klagenfurt, München, Köln und New York City betreut. ANEXIA bietet ihren Kunden hochwertige und individuelle Lösungen im Bereich Web- und Managed Hosting, sowie Individualsoftware und App Entwicklung.

---

prodot GmbH
Schifferstraße 196
47059 Duisburg
T: 0203 - 346945 - 0
F: 0203 - 346945 - 20
info@prodot.de
https://prodot.de

prodot – People. Passion. Performance.
Intelligente Software für internationale Konzerne und mittelständische Unternehmen: prodot stärkt Kunden im weltweiten Wettbewerb – mit effizienten, stabilen und kostensenkenden Lösungen.
Durch das Zusammenspiel aus Know-how, Kreativität und Qualitätsmanagement leisten wir einen Beitrag zum langfristigen Erfolg unserer Auftraggeber. prodot bringt hierzu die richtigen Menschen zusammen.
Seit über 15 Jahren vertrauen uns deshalb Marktführer wie Aldi Süd, Microsoft und Siemens.
Sprechen Sie mich an – als Kunde, Partner oder Kollege. Pascal Kremmers.

eCommerce / Payment

Payone GmbH & Co. KG
Fraunhoferstraße 2-4
24118 Kiel
T: +49 431 25968-400
F: +49 431 25968-1400
sales@payone.de
www.payone.de

PAYONE ist einer der führenden Payment Service Provider und bietet modulare Lösungen zur ganzheitlichen Abwicklung aller Zahlungsprozesse im E-Commerce. Das Leistungsspektrum umfasst die Zahlungsabwicklung von allen relevanten Zahlarten mit integriertem Risikomanagement zur Minimierung von Zahlungsausfällen und Betrug. Standardisierte Schnittstellen und SDKs erlauben eine einfache Integration in bestehende IT- und mobile Systemumgebungen. Über Extensions können auch E-Commerce-Systeme wie Magento, OXID eSales, Demandware, Shopware, plentymarkets und viele weitere unkompliziert angebunden werden.

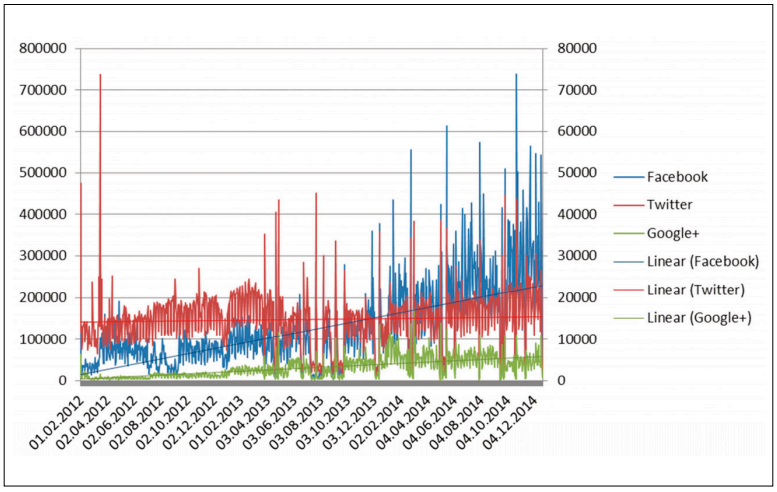
Web- / Mobile-Entwicklung & Content Management

digitalmobil GmbH & Co. KG
Bayerstraße 16a, 80335 München, T: +49 (0) 89 7 41 17 760, info@digitalmobil.com, www.digitalmobil.com

In allen Fragen rund um das Dienstleisterverzeichnis berät Sie Frau Roschke gerne persönlich!
Juliane Roschke ■ 089 / 7 41 17 - 283 ■ juliane.roschke@nmg.de

Die Ausgabe 9/2016 erscheint am 11. August 2016

Deep Integration mit sozialen Netzen



Soziale Netze sind im Web nicht nur allgegenwärtig, sie bieten Entwicklern inzwischen eine Fülle an überaus reizvollen Features. Dank einer durchdachten, tiefen Integration ihrer Webanwendungen und mobiler Apps mit sozialen Netzen können Entwickler sich die Innovationskraft von deren Anbietern zu Eigen machen. Die anhaltende Popularität dieser Dienste verspricht eine langfristige Beständigkeit und hat eine enorme Anziehungskraft für Unternehmen. Aufgrund ihrer beachtlichen Reichweite haben sich soziale Netze in vielseitig nutzbare Marketing-Plattformen verwandelt. Diese Dienste fungieren einerseits als ein Werbemedium im klassischen Sinn, auf der anderen Seite bieten sie Marktforschungsinstrumente, deren Kundennähe den Unternehmen einen deutlichen Mehrwert bietet.

Web Application Security

In vielen Unternehmen bilden webbasierte Applikationen das Herzstück der IT-Infrastruktur. Ob es sich dabei um ein Content-, CRM-, Dokumentenmanagement- oder Workgroup-System handelt, ist zunächst zweitrangig. Sie alle sind für potenzielle Angreifer interessante Ziele. Ein umfassender Schutz ist mehrstufig. EIn Baustein: Mit einem Web Application Security Scanner lassen sich Anwendung auf mögliche Schwachstellen prüfen.

State of the Application

Daten sind das Rückgrat jeder Applikation. Um Daten dauerhaft aufzubewahren, werden sie in ein Speichermedium serialisiert. In real existierenden Anwendungen gibt es aber noch eine andere Art von Daten. Sie enthalten alle notwendigen Informationen, die die Applikation intern benötigt, um ihrer Aufgabe nachzukommen und den aktuellen Zustand zu beschreiben. Diese Informationen fallen in die Kategorie Statusdaten.

Das History API

Jedes Mal, wenn der Browser eine neue Webseite lädt, oder auch, wenn innerhalb einer Webseite zu einer Sprungmarke gesprungen wird, erzeugt er einen neuen Eintrag im Browserverlauf. Über die Eigenschaft *history* des window-Objekts gelangt man an ein Objekt vom Typ *History*, welches diesen Verlauf repräsentiert. Über dieses Objekt kann man Einblick in den Verlauf nehmen und diesen gegebenenfalls auch verändern.

dotnetpro

Ausgabe 8/2016 ab 21. Juli 2016 am Kiosk

Während eine Mensch-Maschinen-Schnittstelle sehr einfach aufgebaut sein kann, erfordert eine Schnittstelle von Maschine zu Mensch einigen Aufwand. Der Bau von Oberflächen kann aufwendig sein.

www.dotnetpro.de

Unsere digitalen Angebote

Wöchentlicher Newsletter
webundmobile.de/newsletter

Shop
shop.webundmobile.de

YouTube
youtube.com/user/developermedia

Facebook
facebook.com/webundmobile

Google +
gplus.to/webundmobile

Twitter
twitter.com/webundmobile

Stellenmarkt

dotnetpro + web & mobile Developer

- 25.800 Exemplare Gesamtauflage
- 25.300 Newsletter-Empfänger
- 66.600 PI'S

.NET Architektur HTML5/JavaScript iOS/Android

Kontakt:
Jens Schmidtmann, Klaus Ahlering • Tel. 089/74117-125 • sales@nmg.de

Ihr Partner für mehr Online-Wachstum

Wir planen, entwickeln und steuern
Websites, **Apps** und **Kampagnen**.

Unsere Ziele sind Ihre Ziele:

- **Mehr Sichtbarkeit**
- **Mehr Traffic**
- **Mehr Leads**
- **Mehr Conversions**

- **Mehr Kunden**

Besuchen Sie uns unter
www.digitalmobil.com

digitalmobil