Templatesyntax von Angular2 S. 32

Alle Neuerungen von Bootstrap 4.0 S. 52

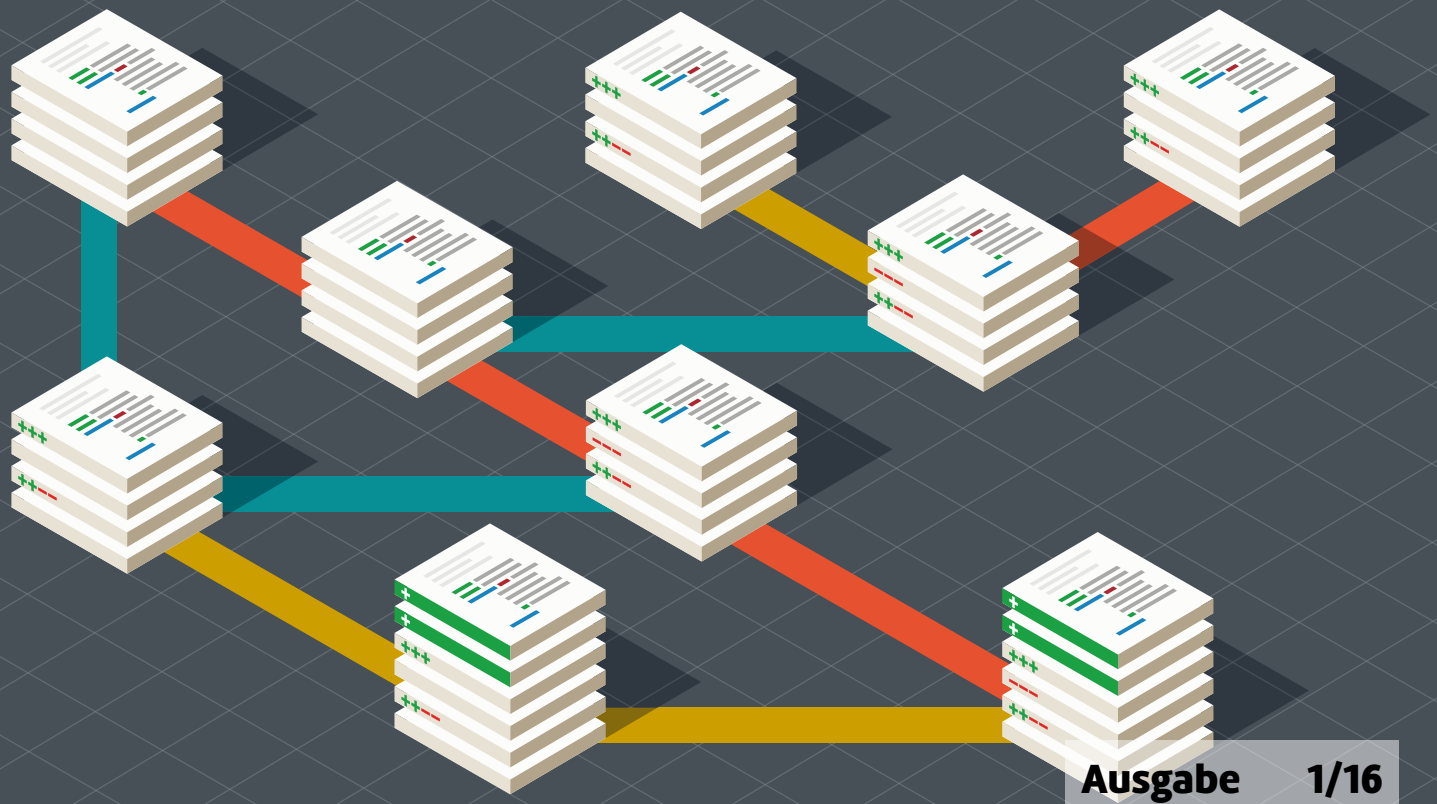
E-Commerce-Software Magento 2 S. 112

web & mobile DEVELOPER

webundmobile.de

Versionen im Griff mit Git

Für professionelle Entwickler ist Version Control System (VCS) ein heißes Thema. ab S. 14



Auf der Heft-CD

Eine Sammlung interessanter JavaScript-Frameworks sowie eine Auswahl leistungsfähiger Tools für Entwickler S. 50

INFO-Programm gemäß § 14 JuSchG

Ausgabe 1/16

Deutschland 14,95 EUR

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

4 198255 014951 01

Version Control

Für professionelle Entwickler, die dazu noch häufig im Team arbeiten, ist VCS ein heißes Thema.

Bei Versionsverwaltungssystemen unterscheidet man prinzipiell zwischen zwei verschiedenen Ansätzen, nämlich zwischen zentralen VCS und dezentralen VCS. Der Nachteil von zentralen VCS ist, dass der entsprechende Server, auf dem das Repository liegt, immer zur Verfügung stehen muss, um eigene Änderungen hochladen oder Aktualisierungen anderer Entwickler aus dem System auf den eigenen Rechner herunterladen zu können. Dieses Problem adressieren dezentrale beziehungsweise verteilte Versionskontrollsysteme wie das im Schwerpunkt dieser Ausgabe vorgestellte Git, das in den vergangenen Jahren in Entwicklerkreisen immer mehr Freunde gefunden hat.

»Für professionelle Entwickler ist Versionsverwaltung ein heißes Thema.«

Das ursprünglich von Twitter ins Leben gerufene Framework Bootstrap ist mittlerweile seit Jahren eines der beliebtesten Repositories auf GitHub und man geht davon aus, dass 13 Prozent aller Websites, die JavaScript verwenden, auch Bootstrap einsetzen. Das sind in etwa 9 Prozent aller Websites im Web. Damit ist Bootstrap nach jQuery die beliebteste JavaScript-Bibliothek überhaupt. Die Macher von Bootstrap wollen diesen Erfolg festigen und haben das schon etwas betagte Framework einer Generalüberholung unterzogen. Das Ergebnis, Bootstrap 4.0, stellt unser Autor Patrick Lobacher ab Seite 52 im Detail vor.

Auch vom bekannten CMS TYPO3 gibt es eine neue Version. Ab Seite 118 erläutert Michael Schams, was für Verbesserungen und Neuerungen TYPO3 CMS 7 LTS zu bieten hat.

Ihr Max Bold
chefredakteur@maxbold.de

Patrick Lobacher

erläutert die neuen Features von Bootstrap 4.0 (S. 52)

Philip Ackermann

präsentiert die Details des Battery Status API (S. 68)

Michael Schams

stellt die neueste Version von TYPO3 vor (S. 118)

INHALT

Aktuell

Microsoft
Zwei Cloud-Rechenzentren in Deutschland geplant 6

Feature

Einführung in die Versionskontrolle mit Git
Das Versionsverwaltungssystem Git unterscheidet sich in vielerlei Hinsicht von anderen Versionsverwaltungssystemen 14

HTML, CSS & JavaScript

Schneller CSS erzeugen mit PostCSS
PostCSS will die etablierten Präprozessoren ablösen 26

Template-Syntax von Angular 2.0
Das Templating mit AngularJS war bereits ein mächtiges Werkzeug. Mit Angular 2.0 legen die Entwickler nun kräftig nach 32

Automatisiertes Testen von Webseiten
Komplexe Webseiten und Single Page Applications benötigen einen hohen Testaufwand 38

Bibliothek Redux
Eine Bibliothek zur Verwaltung von Anwendungszuständen 42

Bootstrap 4.0
Knapp eineinhalb Jahre nach Bootstrap 3 setzt das beliebte Frontend-Framework zum nächsten Versionssprung an 52

Mobile Development

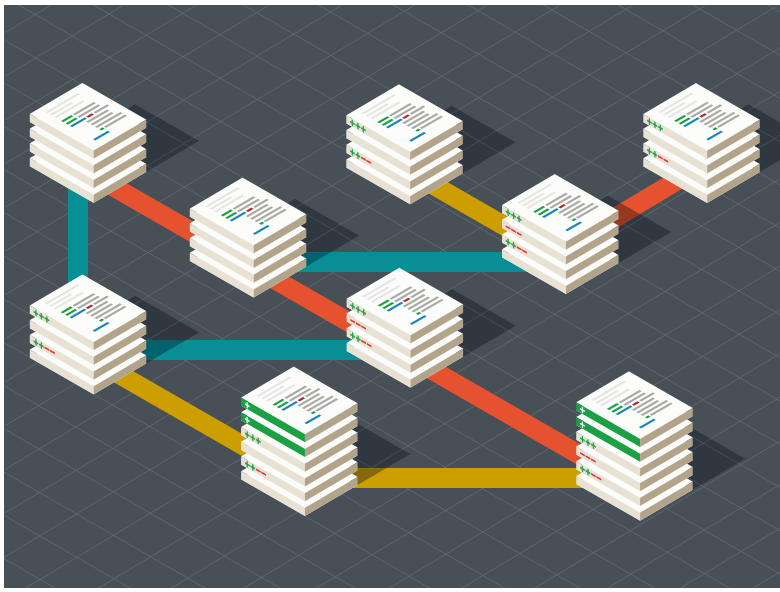
Bluetooth LE
Die Kommunikation zwischen Android und klassischer Hardware erfährt durch Bluetooth LE eine komplette Neugestaltung 58

Battery Status API
In unserer Serie geht es diesmal um das Battery Status API 68

iOS: Core Graphics API
Das Core Graphics API ist eine Schnittstelle zum Zeichnen geometrischer Elemente in einer View 72

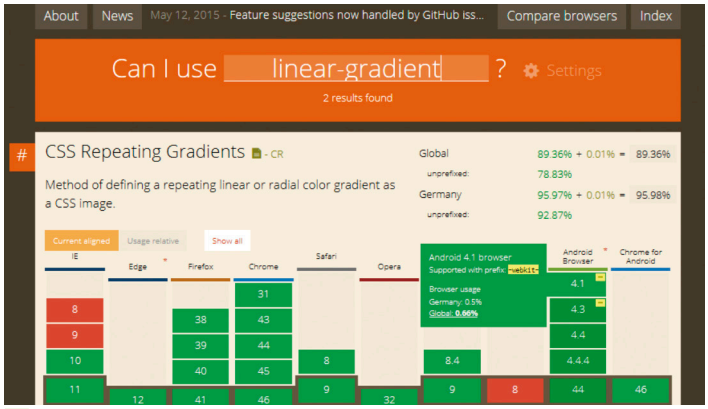
Swift-Performance optimieren
Auch Swift-Code kann in Sachen Performance noch weiter optimiert werden 78

Modularisierte Apps mit Grunt und PhoneGap
Modularisierung ist bei einem umfangreichen Projekt das Gebot der Stunde 82



Mit Versionsverwaltungssystemen kann man bei Softwareprojekten Änderungen am Quelltext protokollieren **S. 14**

Foto: Mathias Vietmeier

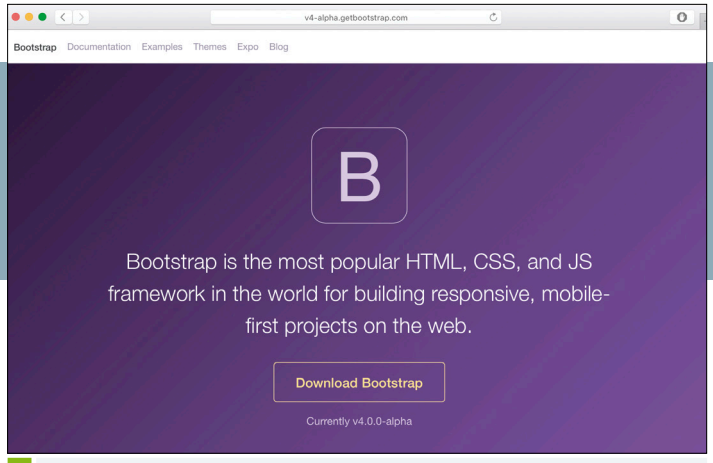


Mit CSS-Präprozessoren und Tools wie PostCSS schreiben Entwickler effektiver und schneller CSS-Code **S. 26**

Experten in dieser Ausgabe

Siegfried Bolz zeigt in einem Artikel, wie man modularisierte Apps mit Grunt und PhoneGap erstellt. 82

Alexander Steireif stellt die neueste Version der Open-Source-E-Commerce-Software Magento vor. 112



Bootstrap sieht sich als Nummer eins der HTML-, CSS- und JavaScript-Frameworks in der Welt **S. 52**

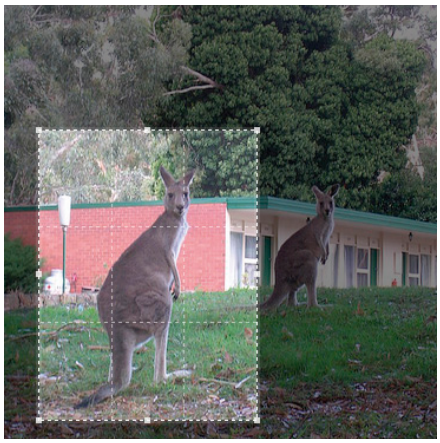
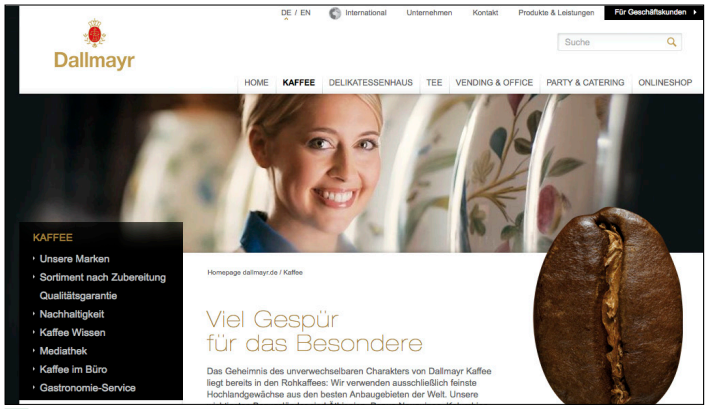


Image Cropping: Mit dieser Funktion lassen sich bestimmte Ausschnitte von Bildern sehr komfortabel im Backend von TYPO3 CMS 7 LTS bearbeiten. **S. 118**



Eine Landingpage soll sich nach einem Klick auf einen Link in einer Suchmaschine oder auf ein Banner öffnen **S. 128**

Xtext, Xtend: Domain-Specific Language für C++ / Qt (Teil 3)
Stressfreie Implementierung dank des Einsatzes einer DSL (Domain-Specific Language) **92**

Strukturhilfe mit Android Studio
Wie Sie mit dem Listview-Control und Android Studio eine hilfreiche App erstellen **98**

Backend

Request-Bibliothek httpful
Immer mehr Systeme müssen miteinander per HTTP-Protokoll kommunizieren **106**

E-Commerce-Software Magento 2
Von der populären E-Commerce Lösung gibt es eine neue und komplett überarbeitete Version **112**

TYPO3 CMS Version 7 LTS
Dieser große Meilenstein in der TYPO3-Geschichte bringt eine Menge an Änderungen und Verbesserungen mit sich **118**

Aimeos E-Commerce-Package
Mit dem E-Commerce-Package Aimeos maßgeschneiderte Webshop-Lösungen in Flow bauen **124**

Beyond Dev

Grafik für Entwickler: Landingpage
Nicht nur der Inhalt, sondern auch der Text- und Bildaufbau bestimmen die Qualität einer Landingpage **128**

Wissensbasierte Applikationen entwickeln mit Ontologien
Semantische Web-Datenbanken mit Web Ontology Language (OWL) – eine Einführung **132**

Standards

Editorial **3**

Heft-CD **50**

Impressum **111**

Online-Recht **140**

Arbeitsmarkt **142**

Dienstleisterverzeichnis **145**

Vorschau **146**

NEWS & TRENDS

AKTUELLE NEWS AUS DER ENTWICKLERSZENE

News

Unternehmen zögern mit digitaler Agenda

Für mehr als zwei Drittel der Unternehmen im DACH-Raum hat die Digitalisierung den Wettbewerb bereits heute verändert. Doch nur 39 Prozent haben schon eine digitale Agenda für sich definiert.

Die Digitalisierung hat den Wettbewerb …

… jetzt schon verändert
68 %

… wird ihn in naher Zukunft verändern (1 – 2 Jahre)
22 %

… wird ihn in fernerer Zukunft verändern (3 – 5 Jahre)
5 %

… wird ihn auf absehbare Zeit nicht verändern
5 %

Hat Ihr Unternehmen bereits eine digitale Agenda?

Ja
39 %

Nein (kommt in den nächsten 12 Monaten)
30 %

Nein (kommt nicht in den nächsten 12 Monaten)
13 %

Nein (kommt nicht in den nächsten fünf Jahren)
18 %

Microsoft

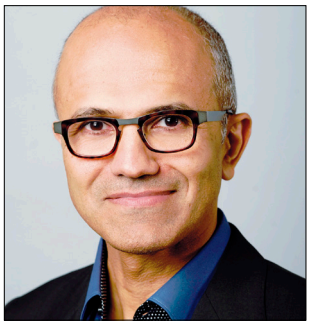
Zwei Cloud-Rechenzentren in Deutschland geplant

Satya Nadella hat in Berlin die neue Microsoft-Cloud-Strategie für Deutschland vorgestellt: Microsoft bietet seine Cloud-Dienste Azure, Office 365 und Dynamics CRM Online zukünftig auch aus deutschen Rechenzentren heraus an.

Mit der neuen Cloud-Lösung will Microsoft sicherstellen, dass Kundendaten ausschließlich innerhalb Deutschlands transportiert und gespeichert werden. Als deutscher Datentreuhänder kontrolliert T-Systems den Zugriff auf die Kundendaten. Als Standorte für die Rechenzentren sind Frankfurt sowie Magdeburg geplant.

Microsoft-CEO Satya Nadella: »Wir wollen jeden Menschen und jede Organisation auf der Welt dazu befähigen, mehr zu erreichen. Die neuen Cloud-Dienste treiben lokale Innovationen und Wachstum voran und bieten Kunden mehr Flexibilität und Wahlmöglichkeiten. Kunden können weiterhin unsere öffentlichen, privaten und hybriden Cloud-Lösungen nutzen oder sich dafür entscheiden, unsere Services aus deutschen Rechenzentren zu beziehen und den Zugang zu ihren Daten durch einen deutschen Datentreuhänder kontrollieren zu lassen«.

»Mit dem neuen Angebot reagieren wir auf die steigende Nachfrage nach unseren Cloud-Diensten in Deutschland. Die Verknüpfung der Innovationskraft und Skalierbarkeit unserer Microsoft-Cloud-Plattform mit deutscher Infrastruktur und deutschem Datentreuhänder ist aus unserer Sicht am Markt einzigartig«, erklärte Alex Stüger, Vorsitzender der Geschäftsführung von Microsoft Deutschland. Wie die Bitkom-Studie



Laut Microsoft-CEO Satya Nadella sollen zwei Cloud-Rechenzentren in Deutschland entstehen

»Cloud Monitor 2015« berichtet, erwarten 83 Prozent der deutschen Unternehmen, dass ihr Cloud-Anbieter seine Rechenzentren ausschließlich in Deutschland betreibt.

Für die neuen deutschen Rechenzentren gelten dieselben Sicherheits-, Service- und Qualitätsstandards wie für alle Rechenzentren von Microsoft. Die neuen Dienste folgen in puncto Sicherheit, Compliance, Transparenz, Datenschutz und Kontrolle denselben Prinzipien wie alle weltweiten Cloud-Services von Microsoft. Die neuen Cloud-Dienste sollen ab der zweiten Jahreshälfte 2016 sukzessive ausgerollt werden und auch Kunden aus anderen europäischen Ländern (EU und EFTA) zur Verfügung stehen.

www.microsoft.com

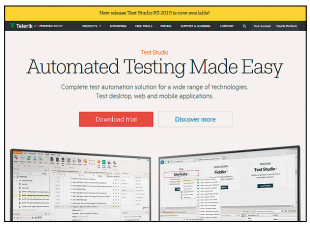
Telerik Test Studio

Native iOS- und Android-Apps automatisch testen

Mit der neuen Lösung Telerik Test Studio Mobile können Tester und Entwickler iOS- und Android-Apps auf echten Mobilgeräten wie auch auf Emulatoren ohne jede Codeprogrammierung automatisiert testen.

Progress hat ein neues Release von Telerik Test Studio, seiner Suite für automatisiertes Testing vorgestellt. Das Telerik Test Studio Mobile, ein neuer Bestandteil der Suite, ermöglicht es jetzt, auch native iOS- und Android-Apps zu testen. Per Point and Click lassen sich komplexe mobile Testfunktionen schnell zusammenstellen und wiederholt anwenden, wodurch die Testprozesse erheblich beschleunigt werden.

Laut einer Studie von Localytics werden 19 Prozent der Apps von US-amerikanischen Nutzern nur ein einziges Mal verwendet. Häufig kehren User einer App den Rücken, weil sie zu kompliziert, zu fehlerhaft oder zu langsam ist. Der permanente Wandel des mobilen Marktes und die hohen Erwartungen der Anwender an die



Vom Telerik Test Studio wurde eine neue Version vorgestellt

Zahl des Monats

Die Fernsteuerung von Rolläden gehört mit **78 %** zu den bekanntesten Smart-Home-Anwendungen bei den deutschen Haus- und Wohnungsbesitzern. Es folgen die Fernsteuerung von Lichtanlagen mit **77 %** und von Heizkörpern beziehungsweise Thermostaten mit **76 %**. Dies sind Ergebnisse einer Umfrage des Marktforschungs- und Beratungsinstituts YouGov.

Quelle: YouGov

Apps setzen die Entwicklungs- und die operativen Teams unter großen Druck. Sie müssen auf Methoden des Rapid Application Development und Deployment setzen, um leistungsfähige mobile Anwendungen schnell zu erstellen.

Um das zu erreichen, müssen die Apps während des Entwicklungsprozesses möglichst oft und möglichst früh getestet werden. Das gilt ganz besonders für agile Teams. Als Low-Code-/No-Code-Lösung konzipiert, ermöglicht Telerik Test Studio Mobile eine enge Zusammenarbeit von Entwicklern und Testern. Sie können gemeinsam stabile und ausgefeilte Apps erstellen, die über verschiedene iOS- und Android-Varianten hinweg zuverlässig und performant laufen.

www.telerik.com/teststudio

Bluetooth-Technologie 2016

Größere Reichweite, höhere Geschwindigkeit und Mesh-Networking

Die Bluetooth Special Interest Group (SIG) hat die Highlights ihrer Technologie-Roadmap für 2016 vorgestellt. In deren Mittelpunkt stehen Erweiterungen der Bluetooth-Funktionalitäten für das Internet der Dinge (Internet of Things, IoT).

Zu den wichtigsten technologischen Updates werden eine größere Reichweite, eine höhere Geschwindigkeit sowie Mesh-Networking zählen. Die Weiterentwicklungen von Bluetooth werden vor allem den schnell wachsenden Branchen wie

Monitoring Report Wirtschaft DIGITAL 2015

Wenig Digitalisierungsdynamik

Laut Wirtschaftsindex DIGITAL, den TNS Infratest und das ZEW im Auftrag des Bundesministeriums für Wirtschaft und Energie veröffentlicht haben, erreicht Deutschland beim Digitalisierungsgrad seiner gewerblichen Wirtschaft gerade einmal 49 von 100 möglichen Indexpunkten.

Der Wirtschaftsindex DIGITAL zeigt, dass sich elf beobachtete Kernbranchen in fünf Digitalisierungsdimensionen zwischen stark überdurchschnittlich bis stark unterdurchschnittlich digitalisiert aufteilen.

Einzig die IKT-Wirtschaft erreicht mit 66 Indexpunkten den höchsten Digitalisierungsgrad und ist damit Vorreiter der digitalen Transformation in Deutschland. Prognostiziert wird diesem Wirtschaftsbereich für 2020 ein Wert von 71 Punkten. Als überdurchschnittlich digitalisiert gelten wissensintensive Dienstleister mit heute 59 und in fünf Jahren 62 Indexpunkten. Finanz- und Versicherungsdienstleister erreichen heute 55 Indexpunkte und 2020 ebenfalls 62 Indexpunkte.

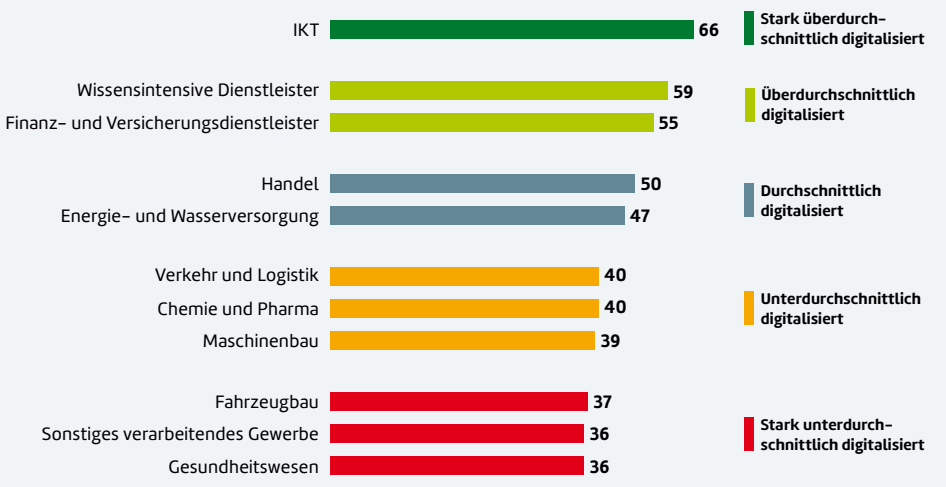
Durchschnittlich digitalisiert zeigt sich mit 50 Indexpunkten im Jahr 2015 der Handel. Prognostiziert wird ihm eine Verbesserung um sechs Punkte. Die Energie- und Wasserversorgung (2015: 47 Punkte) verbessert sich um zwölf Punkte und soll in fünf Jahren 59 Indexpunkte erreichen.

Unterdurchschnittlich digitalisiert sind und bleiben Verkehr und Logistik (2015: 40, 2020: 49 Punkte). Der Maschinenbau zeigt ein hohes Digitalisierungstempo (2015: 39 Punkte, 2020: 51 Punkte) und steigt 2020 in die nächsthöhere Digitalisierungsdimension auf. Dagegen sind die Wirtschaftsbereiche Chemie und Pharma gegenwärtig unterdurchschnittlich digitalisiert (2015: 40 Punkte).

Stark unterdurchschnittlich digitalisiert sind und bleiben die Einrichtungen im deutschen Gesundheitswesen (2015: 36 Punkte, 2020: 44 Indexpunkte). Auch der Fahrzeugbau fällt gegenwärtig mit 37 Punkten in diese Kategorie, steigt aber 2020 mit 48 Punkten in den nächsthöheren Digitalisierungsgrad auf. Ein sehr hohes Digitalisierungstempo hat das sonstige verarbeitende Gewerbe (2015: 36 Punkte, 2020: 50 Punkte), das sich bis 2020 sogar um zwei Digitalisierungsdimensionen zu »durchschnittlich digitalisiert« verbessern wird.

Der Wirtschaftsindex DIGITAL zeigt, wie weit die Digitalisierung in den deutschen Unternehmen aktuell fortgeschritten ist.

Wirtschaftsindex DIGITAL 2015: Branchen-Clustering relativ zur gewerblichen Wirtschaft



Index = 49 Punkte; Repr. Unternehmensbefragung; eig. Berechnung, n = 770

Quelle: TNS Infratest

News

Mobiler Kunden-Support

Support-Lösungen halten bislang nicht Schritt mit der Verbreitung mobiler Anwendungen, findet Citrix. Sobald die App auf dem Smartphone streike, würden herkömmliche Desktop-Programme zur Fernwartung an ihre Grenzen stoßen. Dies möchte Citrix mit zwei Lösungen ändern: Citrix Concierge und GoToAssist Seeit.

Per App können die Nutzer von Citrix Concierge in Kontakt mit dem Support treten und sich helfen lassen. Und mit GoToAssist Seeit können Kunden das Kamerabild ihres Mobilgeräts streamen und so Missverständnisse bei der mündlichen Beschreibung von Problemen vermeiden.

»Organisationen konzentrieren sich immer stärker auf mobile Anwendungen, dennoch bleibt ein intelligenter, schneller und hochwertiger Support für diese Nutzungsart eine Herausforderung. Citrix Concierge und GoToAssist Seeit geben Unternehmen die Möglichkeit, ihren Support mit personalisierten Kundeninteraktionen über Video, Kamera und Chat auf die nächste Ebene zu bringen«, beschreibt Rouven Mayer, Senior Manager bei Citrix, die neuen Lösungen.

www.citrix.de

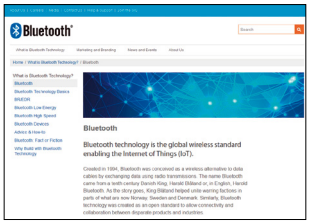
Fotos: Shutterstock / simo988

Smart Home, Industrieautomation, standortbasierte Dienste und intelligente Infrastrukturen zugute kommen.

Von den angekündigten Erweiterungen wird die wachsende Zahl von IoT-Anwendungen profitieren. Die Reichweite von Bluetooth Smart wird sich um das Vierfache erhöhen und so das intelligente Heim sowie Infrastruktur-Anwendungen grundlegend verändern. Zudem wird dadurch eine erweitertete, robuste Verbindung für Full-Home- oder Outdoor-Anwendungen zur Verfügung stehen. Eine Steigerung der Geschwindigkeit um hundert Prozent, ohne dabei den Energieverbrauch zu erhöhen, ermöglicht nicht nur eine schnellere Datenübertragung in kritischen Anwendungen, wie beispielsweise in medizinischen Geräten, sondern erhöht auch die Reaktionsfähigkeit und verkürzt die Latenzzeit. Mit Mesh-Networking sind Bluetooth-fähige Geräte künftig in der Lage, sich miteinander als Netzwerke zu verbinden, die damit ein ganzes Gebäude oder Zuhause abdecken. Dadurch eröffnen sich grundlegend neue Heim- und Industrieautomations-Anwendungen.

»Mit den angekündigten technologischen Erweiterungen berücksichtigen wir die hohe Nachfrage nach neuen Bluetooth-Funktionen durch unsere Mitglieder und durch die Branche als Ganzes«, erläutert Toby Nixon, Chairman of the Bluetooth SIG Board of Directors. »Aktuelle Prognosen gehen bis zum Jahr 2025 von einem Marktpotenzial zwischen 2 und 11,1 Billionen US-Dollar aus. Die technischen Updates von Bluetooth im nächsten Jahr werden dabei helfen, diese Erwartungen umzusetzen und das Wachstum im IoT-Bereich zu beschleunigen.«

www.bluetooth.com



Die Technologie-Roadmap 2016 der Bluetooth SIG dreht sich um das Internet der Dinge

Das Hasso-Plattner-Institut wurde für die neue IT-Sicherheitslösung CloudRAID ausgezeichnet

Hasso-Plattner-Institut (HPI)

Auszeichnung für CloudRAID

Das Hasso-Plattner-Institut (HPI) ist für sein Softwaresystem CloudRAID mit dem Innovationspreis 2015 des IT-Sicherheitsverbands TeleTrust ausgezeichnet worden.

Das Softwaresystem stellt eine Zwischeninstanz zwischen Anwendern und Anbietern von Speicherressourcen in der Cloud dar. Ein erster CloudRAID-Prototyp funktioniert mit Desktop-PCs, Webbrowsern und mobilen Endgeräten (Android und iOS). Alle Dateien eines Nutzers werden in Blöcke aufgespalten und verschlüsselt. Danach werden die Datenpakete auf verschiedene, voneinander unabhängige Dienstleister verteilt. Die Daten können über verschiedene Geräte hinweg synchronisiert sowie mit anderen Nutzern geteilt werden. Dabei verbleibt das Schlüsselmaterial ausschließlich beim Nutzer selbst oder wird auf sichere Weise an autorisierte Nutzer weitergegeben, sodass Speicher- und Dienstbetreiber von CloudRAID selbst niemals auf Inhalte zugreifen können.

RAID-Algorithmen ermöglichen es zudem, flexibel definierbare Anforderungen an Verfügbarkeit und Ausfallsicherheit der Daten umzusetzen. Sowohl öffentliche als auch private Speicher-Anbieter können genutzt werden. Für das Identitätsmanagement sorgt ein unabhängiger Dienst der Bundesdruckerei, der unterschiedlich starke Authentifizierungs-Methoden anbietet – vom Passwort über kryptografische Token bis hin zu Chipkarten wie dem neuen Personalausweis. Anwender bestimmen selbst, wie beim Zugriff auf geteilte Daten die Identität beim Gegenüber nachgewiesen werden soll. Bei besonders vertraulichen Daten wie amtlichen Dokumenten könnte das zum Beispiel nur mit der stärksten Authentifizierung und damit zuverlässigster Verifikation der Identität geschehen.

www.hpi.de

Mobile-Maturity-Studie von Red Hat

Mehr Geld für mobile Applikationen

Laut einer Studie von Red Hat wollen 90 Prozent der Befragten in den nächsten zwölf Monaten ihre Ausgaben für die Entwicklung mobiler Applikationen steigern – um durchschnittlich 24 Prozent.
Die aktuelle Mobile Maturity Survey von Red Hat schließt an die Ergebnisse einer Umfrage von FeedHenry, dem Anbieter einer App-Entwicklungsplattform, an, den Red Hat im Oktober 2014 übernahm.

Im Verlauf ihrer vermehrten Investitionstätigkeit im Mobile-Bereich verfolgen Unternehmen zunehmend einen kooperativen

Studienreihe »Fachkraft 2020«

Erfahrung dank branchennaher Nebenjobs

29 Prozent aller akademischen Absolventen bringen in ihre erste Festanstellung bei Internetunternehmen bereits berufliche Vorerfahrungen aus branchennahen Nebenjobs mit.

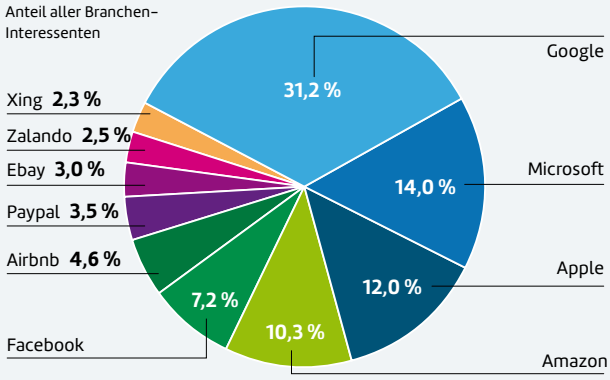
Damit liegt diese Branche, gemeinsam mit dem IT/ Soft-/Hardware-Bereich, auf dem ersten Platz unter 24 analysierten Branchen. Das ist das Ergebnis einer Befragung unter 20.000 Studenten im Rahmen der Studienreihe »Fachkraft 2020« von Studitemps.de und dem Department of Labour Economics der Maastricht University. Bei der Frage nach dem erwarteten Einstiegsgehalt (41.441 Euro) oder der antizipierten Jobzufriedenheit (7,14 von 10) sowie der Sorge vor anfänglicher Arbeitslosigkeit (27 Prozent) erreicht die Internetbranche in der Studie dagegen nur durchschnittliche Ergebnisse.

Top-Wunscharbeitgeber für Studierende im Bereich Internet ist Google: 31,2 Prozent der Studenten würden gerne für den US-Konzern arbeiten. Mit deutlichem Abstand landet Microsoft (14 Prozent) auf dem zweiten Platz, dicht gefolgt vom ewigen Konkurrenten Apple mit 12,0 Prozent. Amazon schafft es mit einem zweistelligen Prozentwert (10,3 Prozent) auf Platz vier der studentischen Wunschliste, Facebook belegt mit 7,2 Prozent den fünften Rang.

Studitemps-Geschäftsführer Eckhard Köhn: »Die branchentypische kreative Aufbruchstimmung bei vielen Internetunternehmen führt zu einer hohen Attraktivität bei Studentinnen und Studenten. Dabei ergeben sich bereits im Studium viele Möglichkeiten, hier beruflich Erfahrungen zu sammeln.

Internetunternehmen profitieren von Studierenden mit viel Berufserfahrung

Die Top-10-Wunscharbeitgeber im Bereich der Internetunternehmen



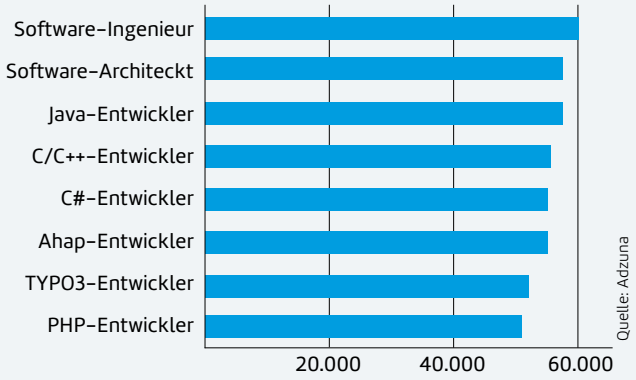
Quelle: Studitemps

Adzuna-Trendreport

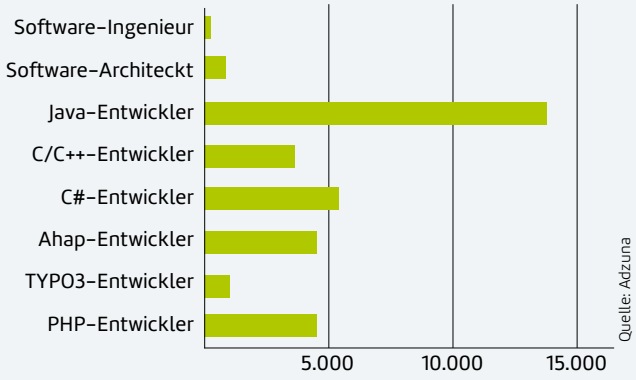
Der Arbeitsmarkt in der IT-Branche

Was sind die wichtigsten Trends auf dem Arbeitsmarkt in der IT-Branche? Wie sehen Chancen und Gehälter aus? Ein Trendreport der Job-Suchmaschine Adzuna gibt einen aktuellen Überblick.
www.adzuna.de

Durchschnittsgehälter

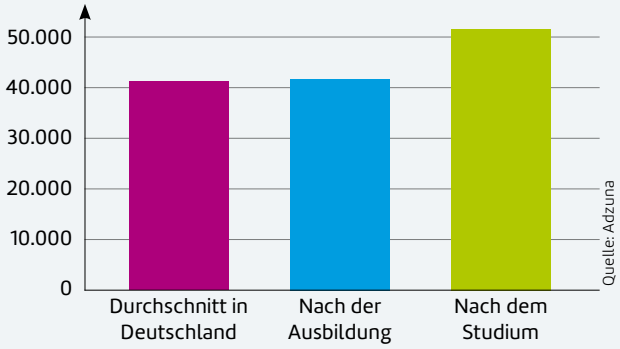


Anzahl offener Stellen



Der Vergleich der Durchschnittsgehälter mit dem Stellenangebot in Deutschland zeigt, dass qualifizierte Java-Entwickler gesucht werden und dass Unternehmen auch bereit sind, diese entsprechend zu bezahlen.

IT-Gehälter im Durchschnitt (€)



Die Gehälter in der IT-Branche sind nicht zyklisch und erleben einen stetigen Anstieg von bis zu 2 Prozent pro Jahr. Dabei liegen die Löhne nur wenig über dem durchschnittlichen deutschen Jahresgehalt.

Ansatz, bei dem die Fachbereiche zusammen mit der IT eine größere Rolle im Entscheidungsprozess spielen. Ein weiterer Trend ist die Einführung neuer, agiler Technologien wie Mobile Backend as a Service (MBaaS) und leichtgewichtiger Programmiersprachen, um die Herausforderungen der mobilen Entwicklung und Integration bewältigen zu können.

Die transformative Kraft von Mobile Computing spielt eine wichtige Rolle für die Unternehmensentwicklung. Ein Drittel der Befragten (35 Prozent) gab an, dass mobile Apps die Unternehmensabläufe ändern, indem Geschäftsprozesse neu gestaltet werden. Weitere 37 Prozent sagten, dass Apps primär zur Automatisierung existierender Prozesse genutzt werden. Ferner machen 24 Prozent der Befragten ihre vorhandenen Webapplikationen mobilfähig und verdeutlichen, dass hier noch weiteres Potenzial vorhanden ist.

Mehr als ein Drittel aller Befragten (37 Prozent) haben ein kollaboratives Mobile Center of Excellence (MCoE) eingerichtet. In mehr als der Hälfte der befragten Unternehmen, die eine Mobile-App-Strategie vollständig implementiert und überprüft haben, gibt es auch ein MCoE. Dies könnte ein Indiz für eine etablierte Mobility-Kultur sein, bei der IT und Fachbereiche optimal kooperieren, um die Produktivität der Mitarbeiter und das Customer Engagement zu fördern.

Unternehmen nutzen Open-Source-Software und MBaaS-Technologie. Eine überwältigende Mehrheit der Befragten (85 Prozent) sagte, dass Open-Source-Software wichtig für ihre Strategie bei der Applikationsentwicklung ist. Außerdem nutzen nahezu ein Drittel der Befragten (31 Prozent) MBaaS-Technologie, um die Herausforderungen der Backend-Integration zu meistern. Deren Anteil wird in den nächsten beiden Jahren voraussichtlich auf 36 Prozent steigen.

Ein neues Zeitalter der leichtgewichtigen Programmiersprachen hat begonnen. Ein Viertel der Befragten (26 Prozent) will in den nächsten beiden Jahren primär Node.js als Programmiersprache für die Backend-Entwicklung nutzen; 15 Prozent setzen vorwiegend

Immer mehr Unternehmen investieren verstärkt in den Mobile-Bereich

auf Java und 19 Prozent auf .NET. Aktuell verwenden 71 Prozent primär Java und 56 Prozent .NET.

Cathal McGloin, Vice President, Mobile Platforms bei Red Hat, kommentierte die Ergebnisse so: »Unternehmen erkennen die Bedeutung einer Mobile-Strategie. Es ist ein klarer Trend in Richtung ausgereifter Praktiken zu erkennen. Beispiele dafür sind die Zusammenarbeit zwischen Fachbereichen und IT, die zunehmende Verbreitung von Open-Source-Software sowie der Einsatz von MBaaS-Technologie und leichtgewichtiger Programmiersprachen. Vor dem Hintergrund der Ergebnisse des Mobile Maturity Survey gehen wir von einem weiteren Wachstum im Bereich der Entwicklung mobiler Applikationen aus. Es gibt in den Unternehmen noch viel zu tun auf diesem Gebiet.«
www.redhat.com

B2B-Vertrieb

Online-Einkauf auf dem Vormarsch

Dank Internetsuche, digitaler Kommunikation und Online-Produktberatung laufen heute 57 Prozent des Einkaufsprozesses online ab.

In Deutschland wie auch in den USA sind fast 50 Prozent der Einkaufsverantwortlichen unter 35 Jahre alt. Das Informations-, Kommunikations- und Beziehungsverhalten dieser Generation unterscheidet sich maßgeblich von dem ihrer Vorgänger: Denn die sogenannten Digital Natives sind es gewohnt, Konsumgüter komfortabel online zu kaufen, und übertragen diese Erfahrungen auf das B2B-Geschäft. Ihr Einkaufsprozess findet daher häufig digital statt. Bis sie schließlich erstmals Kontakt zum Verkäufer aufnehmen, sind so bereits 57 Prozent des Entscheidungsprozesses abgeschlossen.

Foto: Freepik.com

B2B: Auch hier wird verstärkt online gehandelt

»Damit wird die Digitalisierung des Vertriebs zum wichtigen Erfolgsfaktor«, sagt Ralph Lässig, Partner von Roland Berger. »Wer sich nicht an die Bedürfnisse dieser neuen Generation von Einkaufsentscheidern anpasst, setzt langfristig seine Wettbewerbsposition aufs Spiel.«

Die Roland-Berger-Experten haben gemeinsam mit Google eine Umfrage unter knapp 3000 Vertriebsverantwortlichen in B2B-Unternehmen durchgeführt und die Ergebnisse in ihrer Publikation »Die digitale Zukunft des B2B-Vertriebs – Warum Industriegüterunternehmen sich auf veränderte Anforderungen ihrer Kunden einstellen müssen« zusammengefasst.

Demnach sind sich zwar 60 Prozent der Befragten bewusst, dass ein digitaler Vertriebskanal künftig ausschlaggebend für den Geschäftserfolg sein wird.

Doch gerade einmal 42 Prozent der Unternehmen verfolgen auch eine Strategie zum Ausbau digitaler Aktivitäten, und 33 Prozent bieten noch nicht einmal eine Online-Bestellung ihrer Produkte an.

www.rolandberger.com

Dreiländervergleich

Digitale Agenda 2020

Wie weit sind die Firmen der DACH-Region mit der Digitalisierung, wollte die Dreiländer-Studie »Digitale Agenda 2020« von CSC wissen, für die 500 Unternehmensentscheider in ▶

Foto: CSC

Claus Schünemann: Vorsitzender der Geschäftsführung CSC Deutschland

Deutschland, Österreich und der Schweiz befragt wurden.

Offenbar am weitesten in der Planungsphase ist die Schweiz. Dort hat knapp jedes zweite Unternehmen (48 Prozent) bereits eine digitale Agenda verabschiedet. In Österreich sind es 42 Prozent, in Deutschland nur 35 Prozent.

Befragt nach dem Reifegrad ihrer digitalen Projekte, schätzen sich jedoch die deutschen Firmen (37 Prozent) am fortgeschrittensten ein. In Österreich und der Schweiz hält nur rund jedes vierte Unternehmen den eigenen digitalen Reifegrad für hoch bis sehr hoch.

Als größte Stolpersteine für die digitale Transformation sehen die Unternehmen zu wenig Fachkräfte, Finanzierungslücken und Mängel bei der Aus- und Weiterbildung an.

»Die Digitalisierung traditioneller Unternehmen und Behörden fordert grundlegend neue Weichenstellungen für Wettbewerb, Organisation und Kompetenzen«, betont Claus Schünemann, Vorsitzender der Geschäftsführung von CSC Deutschland. »Eine digitale Agenda ist Fundament dafür, mit einer klar definierten Strategie diese revolutionäre Transformation erfolgreich umzusetzen.«

www.csc.com

Foto: SAP

Hartmut Thomsen: Amtierender DsiN-Vorsitzender

Firmen seit dem Jahr 2011. 65 Prozent der Mitarbeiter greifen mittlerweile von außen auf das interne Firmennetzwerk zu. 42 Prozent der Unternehmen nutzen soziale Netzwerke – ein Anstieg um 4 Prozent.

Gleichwohl trifft knapp jeder zehnte Mittelständler keine Schutzvorkehrungen (9 Prozent) und mehr als jeder zweite sichert den E-Mail-Verkehr nicht zusätzlich gegen Fremdzugriff ab (55 Prozent) – 5 Prozent mehr als 2011.

Besonders bedenklich: 73 Prozent der Firmen verzichten laut Studie auf IT-Schulungen ihrer Mitarbeiter und spielen damit dem Social Engineering in die Hände.

Tipps gegen solche Angriffe geben die »Verhaltensregeln zum Social Engineering« von DsiN und Datev.

www.sicher-im-netz.de

Sicherheitsmonitor 2015

IT-Schutz im Mittelstand stagniert

»Der Schutz im Mittelstand vor Cyberangriffen stagniert trotz wachsender Digitalisierung«, bringt Hartmut Thomsen, der amtierende Vorsitzende des Vereins Deutschland sicher im Netz (DsiN), den Sicherheitsmonitor Mittelstand 2015 auf den Punkt.

Er basiert auf kontinuierlichen Befragungen von 7300

Foto: iStock / Jane_Kelly

Cyberangriffe: Schutz im Mittelstand stagniert

Viele Bauchentscheidungen

Ignorierte BI-Tools

In Deutschland ist man vom Ideal des datengetriebenen Unternehmens noch weit entfernt, sagt die Studie »Time is Money« von BARC. Die Analysten befragten 270 Anwender aus dem deutschsprachigen Raum zu ihrer Nutzung von Report- und Analysesoftware.

Ergebnis: Nur 43 Prozent der Entscheidungen beruhen auf einer validen Datenbasis – in kleineren und mittleren Unternehmen sogar nur 37 Prozent. Henrik Jörgensen, Country Manager bei Tableau Software, dem Auftraggeber der Studie, warnt: »Bauchentscheidungen können ihre Berechtigung haben, zum Beispiel wenn es um die Einstellung von Mitarbeitern geht. Entscheidungen, die auf der Basis von Daten getroffen werden, sind aber in der Regel wesentlich tragfähiger und lassen sich auch besser evaluieren.«

Laut BARC haben deutsche Unternehmen allein 2014 fast 1,5 Milliarden Euro für entsprechende Software ausgegeben, aber nur rund 15 Prozent der Mitarbeiter arbeiten damit.

Abhilfe versprechen Self-Service-BI-Tools, mit denen jeder Mitarbeiter Daten sofort analysieren und visualisieren kann. Laut BARC schätzen die Nutzer solcher Tools die Dauer für die Erstellung eines Berichts auf 20 Minuten. Reine Berichtsempfänger dagegen müssten länger als einen Tag auf Ergebnisse warten.

http://barc.de/docs/time-is-money

Mainframe Research Report

Firmen halten dem Mainframe die Treue

BMC Software kommt im 10. Mainframe Research Report zu einem optimistischen Ausblick für Großrechner. Bill Miller, Leiter Solutions Optimization bei BMC, sieht vor allem mobile Services aufgrund ihres Bedarfs an ständiger Verfügbarkeit als Katalysator für das wachsende Interesse an Mainframe-Technologie. Grund dafür sei vor allem deren Fähigkeit zur Bereitstellung geschützter Daten in Kombination mit ihrer hohen Verfügbarkeit.

Hauptsächlich vier Gründe sprechen nach den Aussagen

Foto: IBM

Großrechner im Kommen: IBM LinuxONE Emperor

der 1202 befragten Mainframe-Anwender aus aller Welt dafür, am Mainframe festzuhalten und Investitionen in diese Plattform zu tätigen: 56 Prozent sehen bei Großrechnern Vorteile bezüglich Sicherheit der Zugriffe, 55 Prozent in puncto Verfügbarkeit.

48 Prozent schätzen am Mainframe seine überlegenen Fähigkeiten als zentraler Daten-Server, 45 Prozent freuen sich über den hohen Transaktionsdurchsatz.

Angesichts dieser Stimmungslage ist es kein Wunder, dass 83 Prozent mit stetig steigenden Investitionen in Mainframe-Kapazitäten rechnen.

www.bmc.com

Jetzt kostenlos testen:

www.internetworld.de/probelesen

Die 14-tägige Fachzeitschrift für Digital Professionals!

E-Commerce I Online-Marketing I Technik

Sichern Sie sich jetzt 4 kostenlose Ausgaben der
INTERNET WORLD Business inklusive Business-Newsletter.

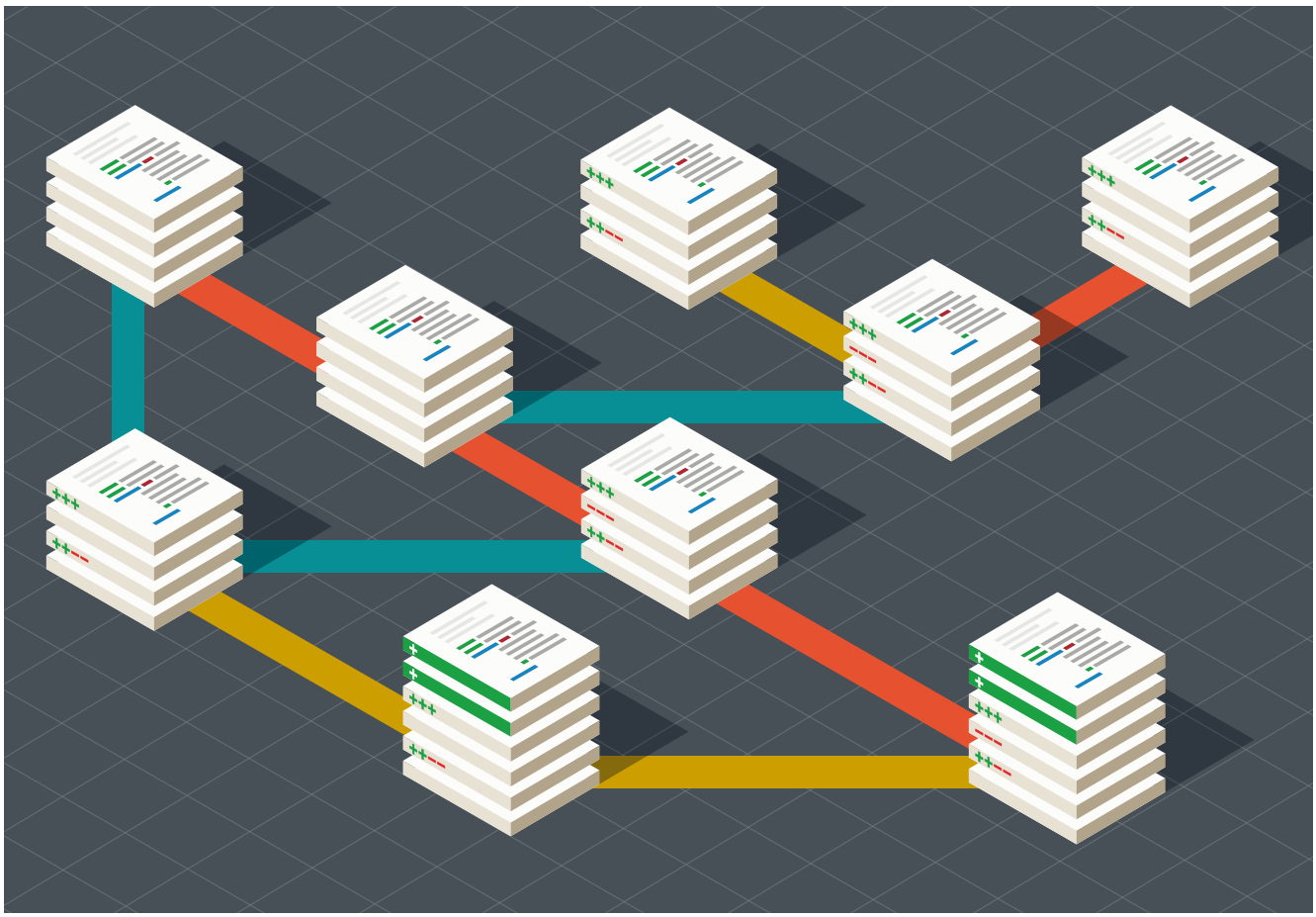


Foto: Mathias Vietmeier

VERSIONSVERWALTUNG MIT GIT

Git kompakt

Das Versionsverwaltungssystem Git unterscheidet sich in vielerlei Hinsicht von anderen Versionsverwaltungssystemen.

Versionsverwaltungssysteme (oft wird auch die Abkürzung VCS für Version Control Systems verwendet) werden innerhalb von Softwareprojekten dazu verwendet, Änderungen am Quelltext über die Zeit hinweg zu protokollieren. Der Vorteil davon dürfte klar sein: Möchte man auf einen alten Zustand des Quelltextes zugreifen, kann man dies bequem über das VCS tun. Zudem kann man Änderungen an Dateien leicht nachvollziehen und notfalls wieder zu alten Zuständen zurückkehren. Und: Für das professionelle Arbeiten im Team sind Versionsverwaltungssysteme quasi unerlässlich.

Bei Versionsverwaltungssystemen unterscheidet man prinzipiell zwischen zwei verschiedenen Ansätzen, nämlich zwischen zentralen VCS und dezentralen VCS. Zentrale VCS waren lange Zeit der Standard in der professionellen Software-Entwicklung. Beispiele hierfür wären CVS oder Subversion (kurz SVN).

Die Idee bei zentralen VCS ist, dass alle versionierten Dateien eines Softwareprojekts innerhalb eines sogenannten Repositorys auf einem zentralen Server verwaltet werden, auf welches dann jeder Entwickler eines Entwicklerteams zugreifen kann. Als Entwickler holt man sich (über entsprechende CVS- oder Subversion-Befehle) die Dateien eines Projekts, mit denen man arbeiten möchte, auf den eigenen Rechner, bearbeitet sie oder fügt neue Dateien hinzu und synchronisiert anschließend (ebenfalls über entsprechende Befehle) wieder mit dem zentralen Server. Bild 1 veranschaulicht diese Vorgehensweise.

Der Nachteil von zentralen VCS ist, dass der entsprechende Server, auf dem das Repository liegt, immer zur Verfügung stehen muss, um eigene Änderungen hochladen oder Aktualisierungen anderer Entwickler aus dem System auf den eigenen Rechner herunterladen zu können. Steht der Server dagegen nicht zur Verfügung, hat man als Entwickler erst

einmal ein Problem. Noch schlimmer ist es, wenn der Server beziehungsweise dessen Festplatte, auf der die gesamten Daten gespeichert sind, kaputt geht.

Dann ist die Historie der Daten nämlich unwiederbringlich verloren – es sei denn, man hat vorher ein Backup gemacht, was in der Regel genau dann nicht der Fall ist. Das Einzige, was in einem solchen Fall bleibt, ist der aktuelle Stand der Daten, den die Entwickler jeweils auf ihrem Rechner haben.

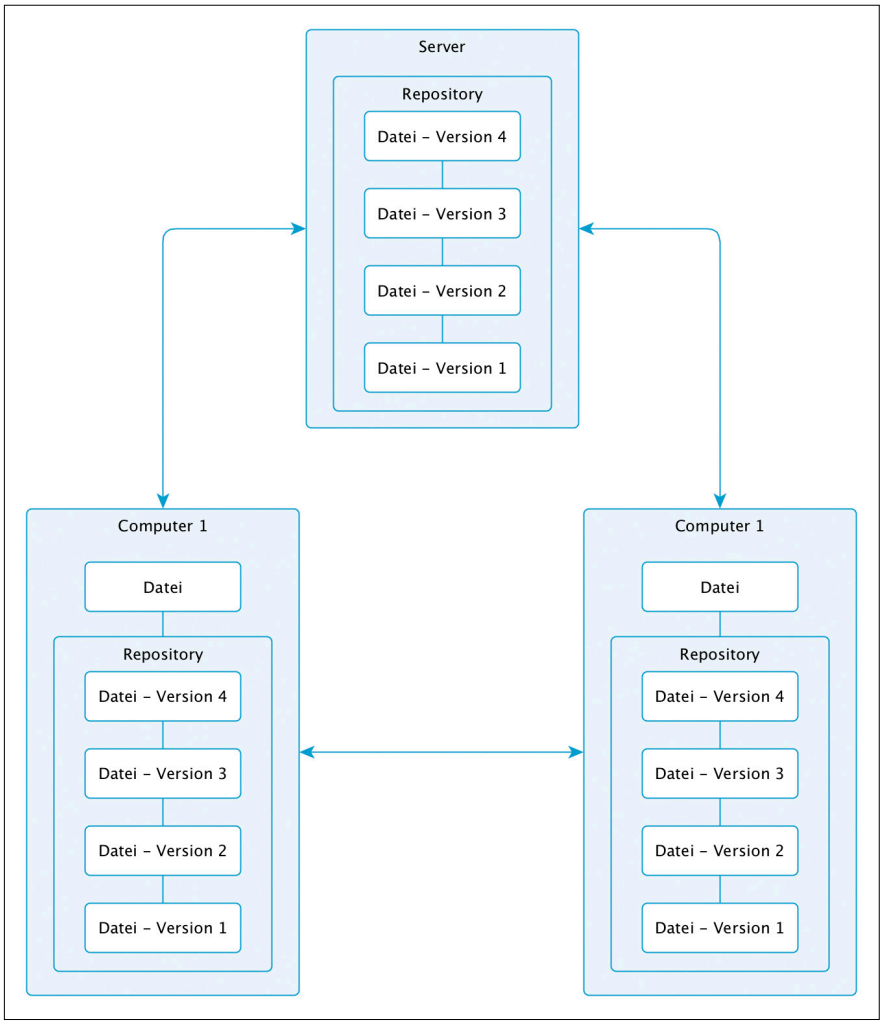
Lokale Repositories

Genau diese beiden angesprochenen Probleme adressieren dezentrale beziehungsweise verteilte Versionskontrollsysteme wie beispielsweise Mercurial oder das im Folgenden vorgestellte und in den vergangenen Jahren immer beliebter gewordene Git.

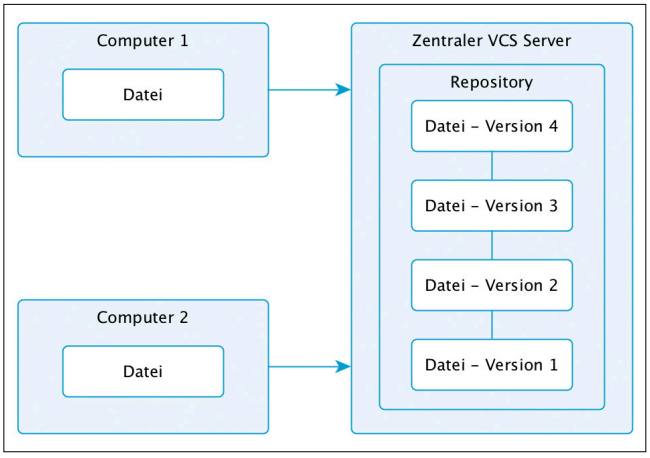
Der Unterschied von dezentralen VCS zu zentralen VCS ist, dass die einzelnen Entwickler nicht mehr nur die letzte Version der Daten auf ihrem Rechner haben, sondern eine vollständige Kopie des gesamten Repositorys (Bild 2), sogenannte lokale Repositories.

Der eine Vorteil: In solche lokalen Repositories können Entwickler Änderungen an den Daten hochladen, ohne mit dem Server verbunden zu sein, und später – wenn wieder eine Verbindung besteht – das lokale Repository mit dem entfernten Repository (dem Remote Repository) synchronisieren.

Der andere Vorteil: Wird der Server in irgendeiner Form beschädigt, sodass die Daten im dortigen (zentralen) Repository verloren gehen, können diese von jedem beliebigen Entwicklerrechner (vorausgesetzt dieser hat den letzten Stand) komplett – das heißt inklusive der gesamten Historie – wiederhergestellt werden.



Dezentral: Bei dezentralisierten Versionskontrollsystemen gibt es mehrere Versionen des Repositorys (Bild 2)



Zentral: Bei zentralisierten Versionskontrollsystemen befindet sich das Repository auf einem zentralen Server (Bild 1)

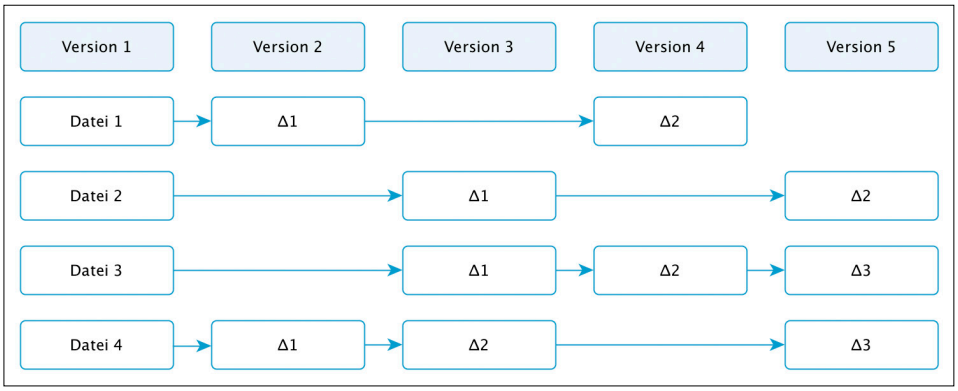
Auch ohne Server

Hinzu kommt, dass die Entwickler auch ohne Server untereinander ihre Repositories synchronisieren können, was bezüglich der Workflows gegenüber zentralen VCS ganz andere Möglichkeiten eröffnet.

Auch die Art und Weise, wie Git Informationen zu den Daten im Repository speichert, unterscheidet sich grundlegend von den meisten anderen Versionskontrollsystemen (unabhängig davon ob zentral oder dezentral).

Während in den meisten anderen VCS nämlich ausgehend von der ersten Version einer Datei alle Informationen als Liste von Änderungen gespeichert werden (in Form sogenannter Deltas), speichert Git bei jedem Commit den kompletten Zustand sämtlicher Dateien des jeweiligen Repositorys (Bild 3).

Git macht also bei jedem Commit einen Snapshot des gesamten Repositorys (Bild 4). Was auf den ersten Blick nach reichlich Speicherverschwendung aussieht, ist in der Pra- ▶



Deltas: Die meisten Versionskontrollsysteme speichern nur die Änderungen (Deltas) an den Daten (Bild 3)

xis relativ speicherschonend. So werden beispielsweise für unveränderte Dateien keine neuen Kopien erzeugt, sondern lediglich Verknüpfungen angelegt.

Ein weiterer Unterschied von Git zu anderen VCS ist die Anzahl an Orten für das Speichern von Daten. In den meisten anderen VCS gibt es zwei Orte dafür: auf der einen Seite die lokale Arbeitskopie (das Working Directory), also die Daten, die man als Entwickler gerade verwendet, sowie auf der anderen Seite das Repository, also die Daten, die bereits übertragen (committet) wurden. In Git dagegen gibt es neben diesen beiden Bereichen noch einen weiteren Speicherort, den sogenannten Staging-Bereich (auch Index oder Staging Area genannt (Bild 5).

Hier stellt man die Änderungen, die man an der Arbeitskopie gemacht hat und in das Repository übertragen (das heißt, committen) möchte, gezielt zusammen.

Arbeitsverzeichnis und Repository

Der Vorteil dieses dritten Bereichs und der damit verbundenen Entkopplung zwischen Arbeitsverzeichnis und dem Repository ist, dass man bei der Zusammenstellung eines Commits viel flexibler ist. Einen Überblick über die wichtigsten Begriffe rund um Git bietet Tabelle 1.

Die Installation von Git ist unabhängig vom verwendetem Betriebssystem relativ einfach. Unter Linux verwendet man je nach Distribution *yum* oder *apt-get*. Sie nutzen also entweder folgenden Befehl für *yum*:

```
$ sudo yum install git
```

Oder Sie verwenden folgenden Befehl im Fall von *apt-get*:

```
$ sudo apt-get install git
```

Für Windows und Mac OS X dagegen können von der Git-Homepage entsprechende Installationsdateien heruntergeladen werden. Die darüber gestarteten Installationsprogramme dürften dann eigentlich selbsterklärend sein.

Nutzernamen und E-Mail-Adresse

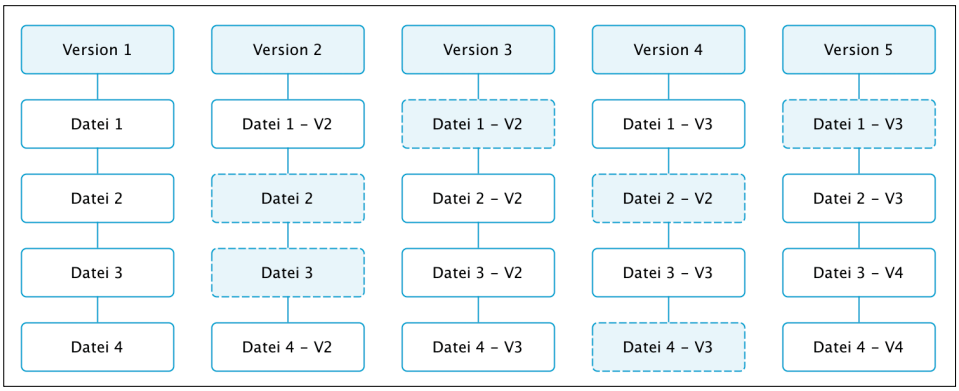
Nach der Installation von Git steht das VCS unter dem Befehl *git* zur Verfügung. Als Erstes sollte man über den Befehl *git config* den Nutzernamen und die E-Mail-Adresse konfigurieren, die standardmäßig mit jedem Commit, den man durchführt, verknüpft werden (bei Bedarf kann beides später auch projektabhängig beziehungsweise für jedes Repository einzeln festgelegt werden):

```
$ git config --global user.name "Max Mustermann"
$ git config --global user.email
"max.mustermann@example.com"
```

Der erste Schritt nach Installation und Konfiguration ist es, ein neues Git-Repository anzulegen. Dies macht man nicht nur einmal, sondern für jedes Projekt, das man über Git versionieren möchte. Um ein neues lokales Git-Repository anzulegen, gibt es prinzipiell zwei Möglichkeiten.

Zum einen kann man ein Verzeichnis auf dem eigenen Rechner als Git-Repository initialisieren, zum anderen kann man ein bestehendes Repository von einem anderen Rechner (beziehungsweise von einem Server) auf den eigenen Rechner herunterladen beziehungsweise klonen. Um ein lokales Verzeichnis als Git-Repository zu initialisieren, führt man den Befehl *git init* aus:

```
$ mkdir example-project
$ cd example-project
$ git init

Initialized empty Git
repository in
/example-project/.git/
```

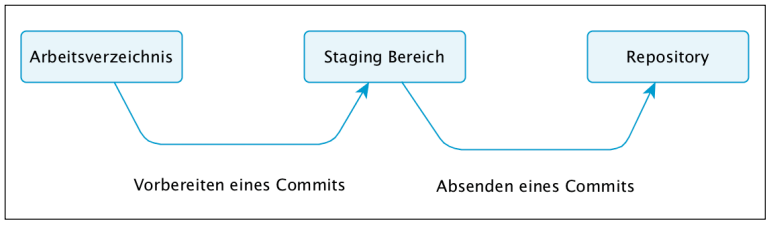


Version: In Git wird bei jedem Commit eine vollständige Version (Snapshot) des gesamten Repositorys gespeichert (Bild 4)

Im Hintergrund wird dadurch im jeweiligen Projektordner (hier:

example-project) das Verzeichnis *.git* erzeugt, in dem Git anschließend alle Informationen über das Repository speichert, beispielsweise die Historie der Dateien und Verzeichnisse im jeweiligen Repository (Bild 6).

Der Befehl *git init* dient lediglich dazu, das entsprechende Verzeichnis als Git-Repository zu initialisieren. Alle eventuell dort vorhandenen Dateien und Unterverzeichnisse werden durch diesen Befehl noch nicht automatisch zum Repository hinzugefügt, sondern befinden sich zunächst noch im eben erwähnten Arbeitsbereich.

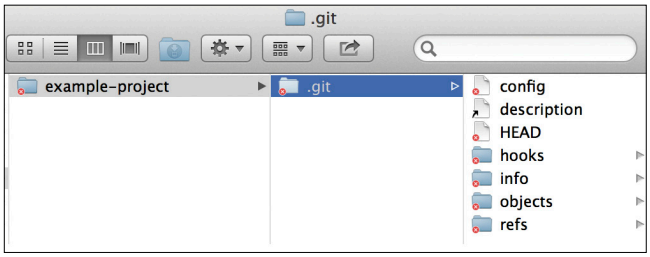


Staging-Bereich: Im Unterschied zu den meisten anderen Versionskontrollsystemen sind Arbeitsbereich und Repository in Git durch den zusätzlichen Staging-Bereich voneinander entkoppelt (Bild 5)

Staging-Bereich

Um diese Dateien zum Repository hinzuzufügen, muss man sie also zuerst in den Staging-Bereich und anschließend ins Repository übertragen. Ersteres, also das Hinzufügen von Dateien zum Staging-Bereich, erreicht man über den Befehl *git add*:

```
$ git add *.js
$ git add **/*.js
$ git add package.json
```



Infos: Im *.git*-Verzeichnis speichert Git alle relevanten Informationen zum Repository (Bild 6)

Wenn man alle Dateien zum Staging-Bereich hinzugefügt hat, die in einem Commit enthalten sein sollen, kann man diese schließlich über den Befehl *git commit* in das lokale Repository übertragen:

```
$ git commit -m "Initial commit"
```

Die Ausgabe lautet dann zum Beispiel wie folgt:

```
[master (root-commit) 671cbc4] Initial commit
  4 files changed, 14 insertions(+)
  create mode 100644 index.js
  create mode 100644 lib/examples.js
  create mode 100644 package.json
  create mode 100644 tests/examples-tests.js
```

Durch dieses zweistufige Übertragen der Daten vom Arbeitsbereich in das Repository ist es möglich, Änderungen an verschiedenen Dateien gezielt zu einem Commit zusammenzufassen und diese gebündelt in das Repository zu übertragen. Wählt man dann noch aussagekräftige Commit Messages, führt dies zu einer übersichtlichen Änderungshistorie.

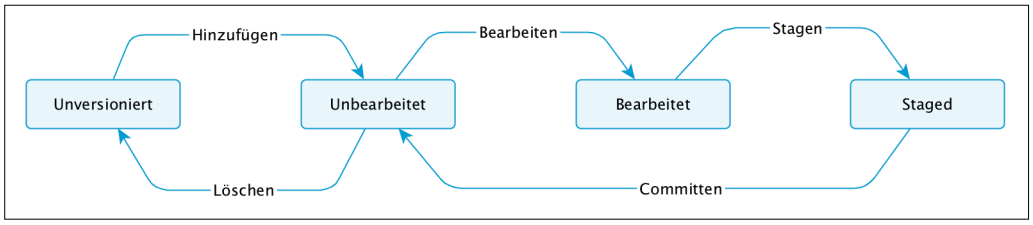
Nach dem Committen befinden sich die Dateien im lokalen Repository, das Arbeitsverzeichnis dagegen enthält keine Änderungen mehr. Überprüfen lässt sich das über den Befehl *git status*, über den sich zu jedem Zeitpunkt nützliche Statusinformationen zum Repository abrufen lassen. Wenn Sie nun den Befehl aufrufen, sollte die Ausgabe wie folgt lauten:

```
$ git status
# On branch master
nothing to commit, working directory clean
```

Mit anderen Worten: Der Arbeitsbereich enthält keine Änderungen, die committet werden könnten. ▶

Tabelle 1: Grundbegriffe in Git

Begriff	Bedeutung
Repository	Enthält alle Dateien eines (Software-) Projekts inklusive vorausgegangener Versionen
Remote Repository	Entferntes Repository
Working Directory	Arbeitsbereich
Index / Staging Area	Bereich, in dem Änderungen zum Übertragen in das lokale Repository vorbereitet werden
Commit	Übertragen der Änderungen aus dem Staging-Bereich in das lokale Repository
Push	Übertragen der in das lokale Repository übertragenen Änderungen in das Remote Repository
Pull	Übertragen der Änderungen aus dem Remote Repository in das lokale Repository
Checkout	Holen einer Arbeitskopie aus einem Branch oder einem Commit
Clone	Kopieren eines Remote Repositorys
Branch	Ein separater Entwicklungszweig
Merge	Zusammenfügen von Änderungen von einem Branch in einen anderen Branch
Fork	Ableger eines Repositorys
HEAD	Aktueller Branch



Dateien können in Git verschiedene Zustände einnehmen (Bild 7)

Dateien können in Git einen von mehreren Zuständen annehmen: untracked (beziehungsweise unversioniert) bedeutet, dass eine Datei noch gar nicht von der Versionskontrolle berücksichtigt wird, modified (beziehungsweise bearbeitet) bedeutet, dass die Datei seit dem letzten Commit geändert wurde, unmodified (beziehungsweise unbearbeitet) dagegen, dass sie seit dem letzten Commit nicht verändert wurde und staged, dass eine geänderte Datei für den nächsten Commit vorgesehen ist (Bild 7).

Ändert man nun Dateien in der Arbeitskopie oder fügt man neue Dateien hinzu, dann kann man über *git status* herausfinden, welche Dateien davon betroffen sind und welchen Status diese Dateien haben:

```
$ git status
# On branch master
# Changes not staged for commit:
# (use "git add <file>..." to update what will be
  committed)
# (use "git checkout -- <file>..." to discard changes in
  working directory)
#   modified:   lib/examples.js
# Untracked files:
# (use "git add <file>..." to include in what will be
  committed)
#   lib/examples2.js
#   tests/examples2-tests.js
no changes added to commit (use "git add" and/or
"git commit -a")
```

Die obige Ausgabe besagt beispielsweise, dass eine Datei verändert wurde *(lib/examples.js)* und dass zwei Dateien als untracked erkannt wurden, sprich bisher nicht versioniert werden *(lib/examples2.js* und *tests/examples2-tests.js)*. Außerdem ist erkennbar, dass die Änderungen an der geänderten Datei sich nur im Arbeitsbereich, nicht aber im Staging-Bereich befinden (zu erkennen an der Ausgabe *Changes not staged for commit)*. Über *git add* und *git commit* könnte man die Änderungen nun wieder in das Repository übertragen.

Unabhängig davon befinden sich allerdings alle Commits bisher nur in dem lokalen Repository. Der nächste Schritt besteht also darin, ein entferntes Repository (das Remote Repository) einzurichten. Hat man dagegen ein bereits existierendes Remote Repository über den Befehl *git clone* auf den eigenen Rechner geklont, entfällt dieser Schritt natürlich, denn dann steht das Remote Repository bereits fest.

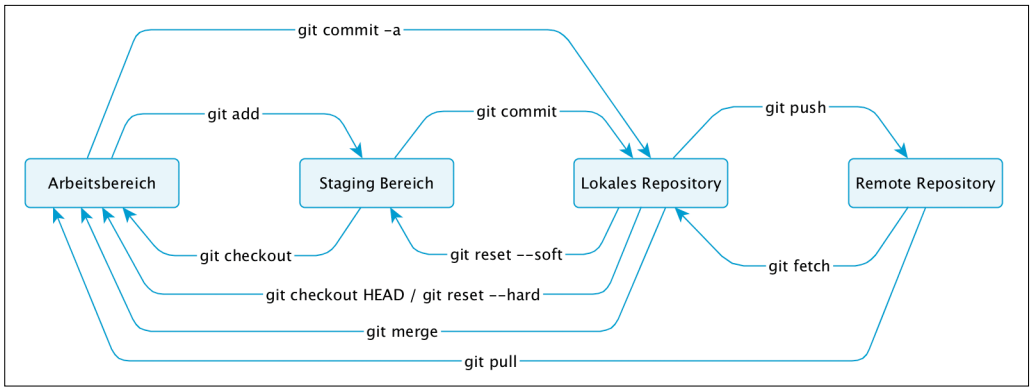
Ein neues Remote Repository lässt sich über den Befehl *git remote add* hinzufügen:

```
$ git remote add origin ssh://cleancoderocker@workspace/
volume1/repository/example-project.git/
```

Der erste Parameter (im Beispiel *origin)* bezeichnet den Kurznamen des Repositorys, über das sich das Repository anschließend bei der Formulierung von Git-Befehlen ansprechen lässt. Der zweite Parameter bezeichnet den URL des Remote Repositorys.

Remote Repository

Um nun die Commits aus dem lokalen Repository in das Remote Repository zu übertragen, verwendet man den Befehl *git push.* Diesem Befehl übergibt man als ersten Parameter den Namen des Remote Repositorys (denn prinzipiell können mit einem lokalen Repository auch mehrere Remote Repositories verknüpft sein) sowie als zweiten Parameter den Branch, aus dem die Commits übertragen werden sollen. Standardmäßig handelt es sich dabei um den sogenannten Master-Branch:



Kommandos: Die verschiedenen Git-Kommandos in der Übersicht (Bild 8)

```
$ git push origin
master
Counting objects: 12,
done.
Delta compression using
up to 4 threads.
Compressing objects:
100% (9/9), done.
Writing objects:
100% (12/12), 1.05 KiB
| 0 bytes/s, done.
Total 12 (delta 1),
reused 0 (delta 0)
```

```
To ssh://cleancoderocker@workspace/volume1/repository/
example-project.git/
 * [new branch]      master -> master
```

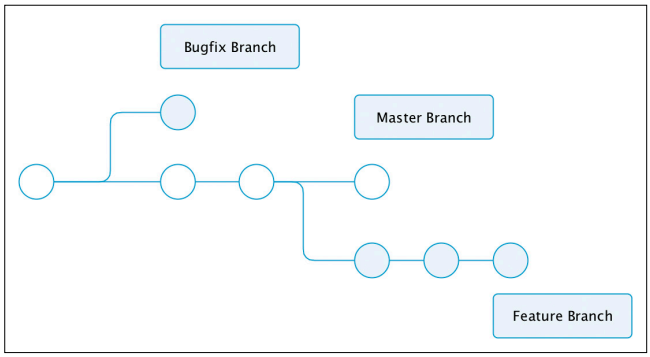
Wenn man im Team arbeitet, dann ist es in der Regel so, dass man, bevor man seine eigenen Änderungen (beziehungsweise Commits) in das Remote Repository pusht, die Änderungen anderer Entwickler aus dem Remote Repository auf den eigenen Rechner beziehungsweise in das lokale Repository überträgt. Dies erledigt man über den Befehl *git fetch*, dem man als Parameter den Namen des Remote Repositorys übergibt:

```
$ git fetch origin
```

▶

Tabelle 2: Die wichtigsten Git-Kommandos

Befehl	Beschreibung
Erzeugen eines Repositorys	
git init	Initialisieren eines neuen Git-Repositorys
git clone ssh://<username>@<domain>/<repositoryname>.git	Klonen eines bestehenden Repositorys auf Basis des übergebenen URL
git clone git@github.com:<username>/<repositoryname>	Klonen eines bestehenden Repositorys auf Basis des übergebenen GitHub-URL
Lokale Änderungen	
git status	Ausgabe von Statusinformationen
git diff	Anzeigen von Änderungen der versionierten Dateien
git add .	Hinzufügen aller lokalen Änderungen zum nächsten Commit
git add -p DATEI	Hinzufügen der Änderungen an einer Datei zum nächsten Commit
git commit -a	Commit aller getrackten Änderungen
git commit -m "Update"	Commit der Änderungen aus dem Staging-Bereich
git commit --amend	Ändern des letzten Commits (Achtung: keine Commits ändern, die bereits in ein Remote Repository hochgeladen wurden)
Commit-Historie	
git log	Ausgabe aller Commits, beginnend mit dem letzten Commit
git log -p <file>	Ausgabe aller Commits für eine Datei, beginnend mit dem letzten Commit
git blame <file>	Ausgabe, wer wann welche Änderungen an einer Datei vorgenommen hat
git stash	Übertragen der lokalen Änderungen aus dem Arbeitsbereich in Zwischenspeicher
git stash pop	Übertragen der lokalen Änderungen aus Zwischenspeicher in den Arbeitsbereich
Branches und Tags	
git branch -av	Auflisten aller Branches
git checkout <branch>	Wechseln in einen bestehenden Branch
git branch <branch>	Anlegen eines neuen lokalen Branchs
git checkout --track <remote>/<branch>	Anlegen eines neuen lokalen Branchs auf Basis eines Remote Branchs
git branch -d <branch>	Löschen eines lokalen Branchs
git tag <tagname>	Hinzufügen eines Tags zum aktuellen Commit
Update und Publish	
git remote -v	Auflisten aller Remote Repositories
git remote show <remote>	Anzeigen der Informationen zu einem bestimmten Remote Repository
git remote add <shortname> <url>	Hinzufügen eines neuen Remote Repositorys
git fetch <remote>	Herunterladen aller Änderungen vom Remote Repository, aber keine Integration in HEAD
git pull <remote> <branch>	Herunterladen aller Änderungen vom Remote Repository und direkte Integration beziehungsweise Merging in HEAD
git push <remote> <branch>	Hochladen der lokalen Änderungen zum Remote Repository
git branch -dr <remote>/<branch>	Löschen eines Branches im Remote Repository
git push --tags	Hochladen der Tags zum Remote Repository
Merge und Rebase	
git merge <branch>	Merging der Änderungen aus einem Branch in den aktuellen HEAD
git rebase <branch>	Rebase des aktuellen HEADs auf den angegebenen Branch
Undo	
git reset --hard HEAD	Verwerfen aller lokalen Änderungen im Arbeitsbereich
git checkout HEAD <file>	Verwerfen der lokalen Änderungen an einer einzelnen Datei im Arbeitsbereich
git revert <commit>	Rückgängigmachen eines Commits
git reset --hard <commit>	Zurücksetzen des HEADs auf einen früheren Commit und Verwerfen aller seitdem stattgefundenen lokalen Änderungen
git reset <commit>	Zurücksetzen des HEADs auf einen früheren Commit und Behalten aller seitdem stattgefundenen lokalen Änderungen
git reset --keep <commit>	Zurücksetzen des HEADs auf einen früheren Commit und Behalten aller seitdem stattgefundenen lokalen Änderungen, die noch nicht committet wurden



Branches: Über separate Branches lassen sich neue Features und Bugfixing vom Hauptentwicklungszweig abkapseln (Bild 9)

Dadurch werden alle Commits aus dem Remote Repository, die sich noch nicht im lokalen Repository befinden, genau dorthin übertragen, und zwar in sogenannte Remote Branches, sprich Entwicklungszweige, die parallel zu dem Hauptentwicklungszweig existieren.

Auf diese Weise hat man Gelegenheit, entsprechende Änderungen zu begutachten, bevor man sie anschließend in den eigenen Master-Branch übernimmt. Letzteres wiederum geschieht über den Befehl *git merge*, dem der Name des Remote Branchs als Parameter übergeben wird:

```
$ git merge origin/master
```

Hierbei kann es passieren, dass Änderungen anderer Entwickler Konflikte mit den eigenen Änderungen auslösen. Diese können dann per Hand über die Kommandozeile oder über den jeweils verwendeten Editor oder die jeweils verwendete Entwicklungsumgebung gelöst werden. Alternativ zu der Kombination aus *git fetch* und anschließendem *git merge* kann man auch den Befehl *git pull* verwenden, der quasi implizit beide Schritte hintereinander ausführt.

Bild 8 fasst noch einmal die wichtigsten Befehle rund um das Arbeiten mit Git zusammen und zeigt den Zusammenhang zu den einzelnen Git-Bereichen. Eine detaillierte Übersicht über eine Auswahl in der Praxis gebräuchlicher Befehle finden Sie zudem in Tabelle 2.

Unabhängige Entwicklungszweige

Ein wesentlicher Bestandteil bei der Arbeit mit Git sind sogenannte Branches. Dabei handelt es sich um unabhängige Entwicklungszweige, über die es möglich ist, unabhängig vom Hauptentwicklungszweig neue Features zu entwickeln oder Bugfixing zu betreiben. Standardmäßig arbeitet man wie bereits gesagt auf dem Master-Branch (der durch den Befehl *git init* standardmäßig angelegt wird).

Über den Befehl *git branch* lassen sich alle diejenigen Branches eines Repositorys auflisten, welches momentan für das Beispiel lediglich einen Branch, nämlich den Master-Branch, auflistet:

```
$ git branch
* master
```

Neue Branches können dagegen über den Befehl *git branch BRANCH* erzeugt werden, wobei für *BRANCH* der Name des neuen Branches einzusetzen ist.

```
$ git branch feature-xyz
$ git branch
feature-xyz
* master
```

Branches sind in Git ein wichtiger Bestandteil für den Entwicklungsprozess: Möchte man beispielsweise ein neues Feature hinzufügen oder einen Fehler beseitigen (Stichwort Bugfixing), so erzeugt man in der Regel einen neuen Branch, um die jeweils damit verbundenen Änderungen von dem Quelltext des Hauptentwicklungszweigs abzukapseln. Auf diese Weise stellt man sicher, dass instabiler Code nicht direkt in den Hautpentwicklungszweig übertragen wird.

Bild 9 veranschaulicht diese Vorgehensweise. Neben dem Master-Branch sind hier zwei isolierte Entwicklungslinien zu sehen: zum einen ein Feature-Branch, zum anderen ein Bugfix-Branch.

Um einen Branch zu wechseln, verwendet man den Befehl *git checkout BRANCH*, wobei *BRANCH* wieder für den Namen des Branchs steht, in den man wechseln möchte. Um beispielsweise in den eben angelegten Branch *feature-xyz* zu wechseln, schreibt man Folgendes:

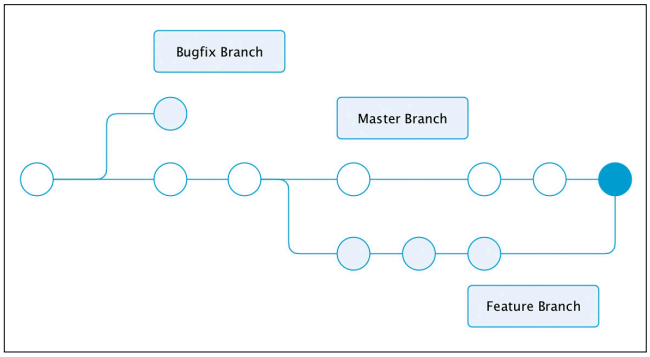
```
$ git checkout feature-xyz
```

Anschließend erhält man als Bestätigung die Meldung *Switched to branch feature-xyz.*

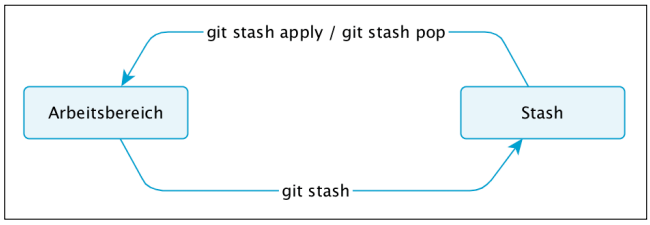
Dass man sich in dem neuen Branch befindet, kann man auch jederzeit über den Befehl *git branch* überprüfen: Der aktive Branch ist in der anschließenden Ausgabe mit einem *-Symbol gekennzeichnet:

```
$ git branch
* feature-xyz
master
```

Ist man mit einem neuen Feature oder dem Beheben eines Bugs im entsprechenden Branch fertig und möchte die damit



Merge: Änderungen aus den Branches lassen sich über *git merge* wieder in den Hauptentwicklungszweig übertragen (Bild 10)



Stash: Über den Stash lassen sich Änderungen am Arbeitsbereich zwischenspeichern (Bild 11)

verbundenen Änderungen in den Hauptentwicklungszweig übernehmen, wechselt man zunächst in den Branch, in den die Änderungen übernommen werden sollen, und verwendet anschließend den Befehl *git merge*:

```
$ git checkout master
$ git merge feature-xyz
Merge made by the 'recursive' strategy.
  lib/examples.js | 2 ++
  1 file changed, 2 insertions(+)
```

Im anschließend erscheinenden Kommandozeilendialog hat man die Möglichkeit, die Commit-Meldung anzupassen – standardmäßig *Merge branch BRANCH*, wobei *BRANCH* wieder für den Branchnamen steht. Bild 10 veranschaulicht das Prinzip.

Stashing

Wenn man mit mehreren Branches arbeitet, kann es bei der Entwicklung durchaus sein, dass man immer wieder zwischen verschiedenen Branches hin- und herwechseln muss, beispielsweise weil man gerade an zwei verschiedenen Features arbeitet. Oft hat man dann den Fall, dass man von dem einen in den anderen Branch wechseln möchte, allerdings im jeweiligen Arbeitsbereich bereits Änderungen gemacht hat und diese – weil sie eventuell noch nicht fertig sind – noch nicht committen möchte.

Dann steht man vor einem Problem: Würde man nämlich trotz der Änderungen im Arbeitsbereich in den anderen Branch wechseln, gingen die Änderungen verloren. Doch Git bietet diesbezüglich eine Lösung an: den sogenannten Stash, eine Art Zwischenspeicher, in dem genau zu oben genanntem Zweck Änderungen an den Daten zwischengespeichert und später wieder abgerufen werden können (Bild 11).

Um alle lokalen Änderungen im Arbeitsbereich in diesen Zwischenspeicher zu bewegen, reicht die Eingabe des Befehls *git stash*:

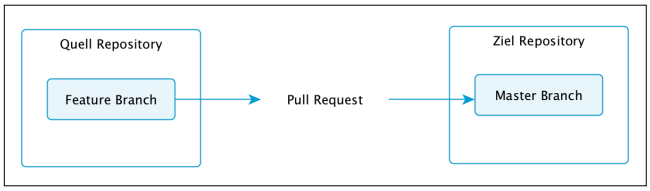
```
$ git stash
Saved working directory and index state WIP on master:
f141859 Merge branch 'feature-xyz'
HEAD is now at f141859 Merge branch 'feature-xyz'
```

Anschließend ist der jeweilige Arbeitsbereich wieder frei von Änderungen und es kann problemlos in einen anderen Branch gewechselt werden.

Der Befehl *git stash* lässt sich – Änderungen im Arbeitsbereich vorausgesetzt – auch mehrmals anwenden. Die jeweiligen Änderungen werden dann – wie bei einem Stack – aufeinander gespeichert, sprich die zuerst gestashten Änderungen liegen unten in diesem Stack, die zuletzt gestashten Änderungen ganz oben. Über den Befehl *git stash list* lassen sich alle Änderungen des Zwischenspeichers ausgeben:

```
$ git stash list
stash@{0}: WIP on master: f141859 Merge branch
'feature-xyz'
stash@{1}: WIP on master: c149030 Added something else
stash@{2}: WIP on master: f401859 Added something
stash@{3}: WIP on master: d231880 Added logger
stash@{4}: WIP on master: f241408 Added toString()
method
```

Über *git stash apply* können die dem Stash zuletzt hinzugefügten Änderungen wieder auf den Arbeitsbereich angewandt werden. Das Gleiche erreicht man auch über den Befehl *git stash pop*, wobei hier die jeweiligen Änderungen auch direkt vom Stash entfernt werden.



Pull Requests stellen Anfragen zur Integration von Code dar (Bild 12)

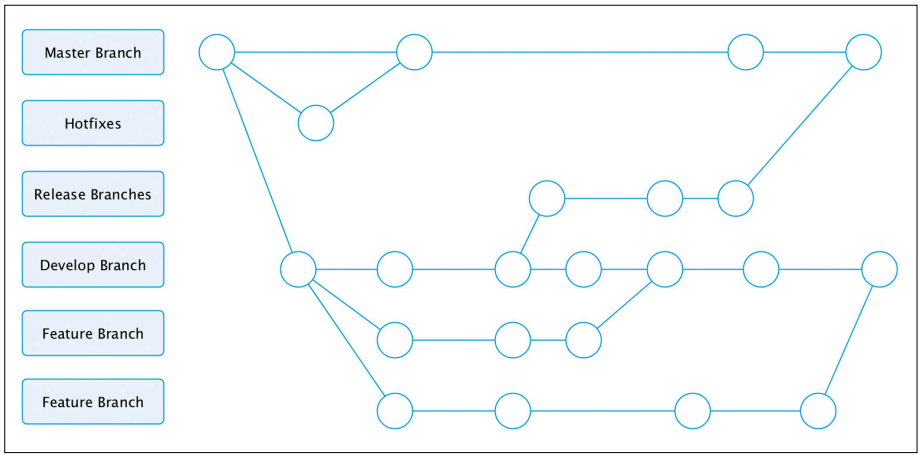
Workflows

Wie erwähnt eröffnet Git aufgrund seiner dezentralen Struktur ganz neue Herangehensweisen beziehungsweise ganz neue Workflows bezüglich der Arbeit im Team.

Der einfachste Workflow ist der sogenannte zentralisierte Workflow. Er entspricht quasi dem Gedanken von zentralen Versionsverwaltungssystemen am ehesten: Hierbei gibt es, wie auch beispielsweise bei Subversion, einen zentralen Server mit dem Hauptrepository, das die einzelnen Entwickler im Team auf ihren jeweiligen Rechner klonen und somit jeweils ihr eigenes lokales Repository erzeugen.

In diesem lokalen Repository nehmen sie dann wie oben beschrieben Änderungen vor und übertragen diese nach und nach (über *git push)* in das Hauptrepository. In der Regel braucht man für diesen zentralisierten und zugleich recht einfachen Workflow abgesehen von dem Master-Branch keinen weiteren Branch.

Als Ausbaustufe des zentralisierten Workflows fungiert der sogenannte Feature-Branch-Workflow. Dabei werden einzelne Features in separaten Branches entwickelt und bei Fertigstellung des jeweiligen Features wieder in den Master-Branch übertragen. ▶



Workflow-Variante: Der Gitflow-Workflow (Bild 13)

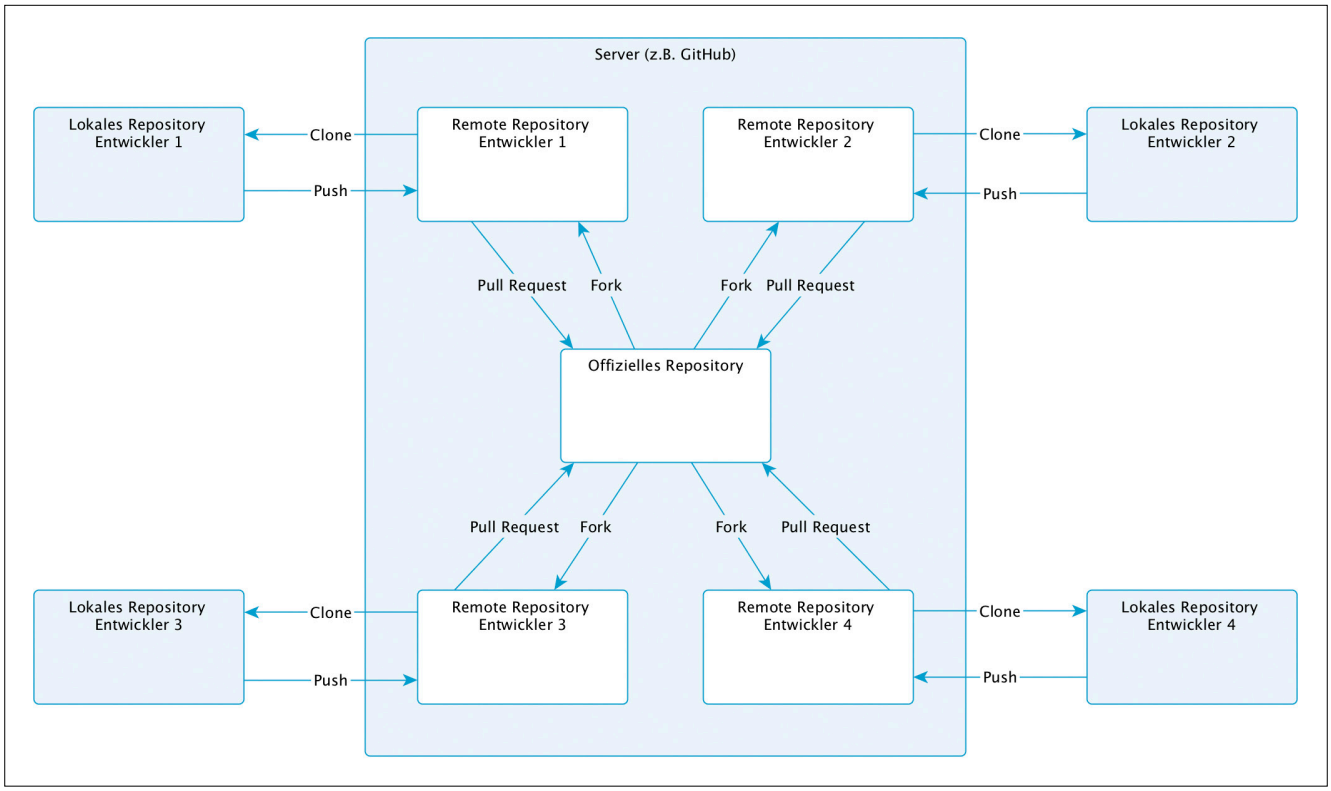
Möchte man fertiggestellte Features zunächst im Team besprechen, bietet sich außerdem die Verwendung sogenannter Pull Requests an. Dabei handelt es sich um Mitteilungen eines Entwicklers an die anderen Entwickler im Team, dass er ein bestimmtes Feature fertiggestellt hat und nun quasi darum bittet, die damit verbundenen Änderungen in das zentrale Repository zu übertragen. Andere Entwickler im Team haben durch den Pull Request die Möglichkeit, die Änderungen zu begutachten oder zu kommentieren (Bild 12).

Prinzipiell sind Pull Requests unabhängig von dem Feature-Branch-Workflow und können beispielsweise auch bei anderen – im Folgenden besprochenen – Workflows verwendet werden. Beim zentralisierten Workflow dagegen können Pull Requests nicht verwendet werden, da hier die Voraussetzung dafür – sprich zwei verschiedene Repositories – nicht gegeben ist.

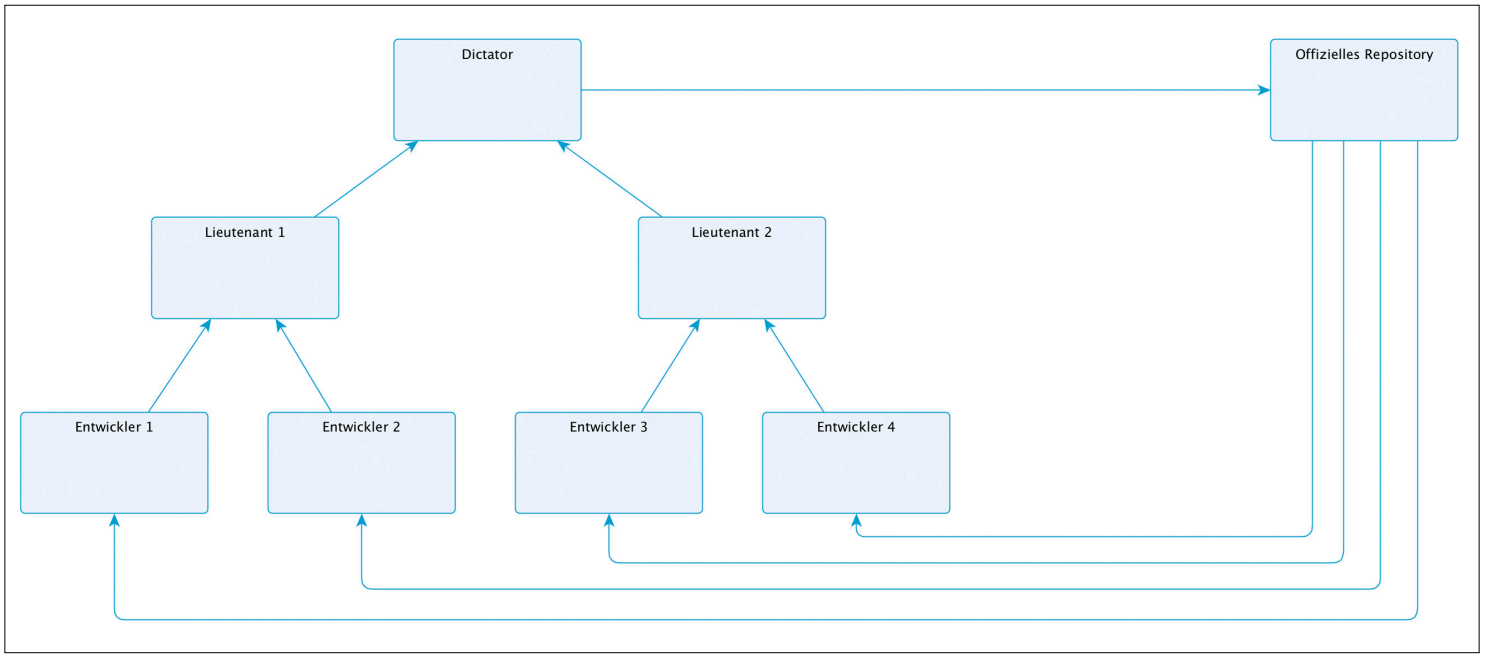
Wie der zentralisierte Workflow ist auch der Feature-Branch-Workflow noch relativ einfach gehalten. Deutlich komplexer wird es dagegen mit dem sogenannten Gitflow-Workflow (http://nvie.com/posts/a-successful-git-branching-model), der vor allem dann sinnvoll ist, wenn in regelmäßigen Abständen Releases einer Software ausgeliefert werden sollen (Bild 13).

Die grundlegenden Abläufe wie das Arbeiten im lokalen Repository, Pull Requests und so fort bleiben mehr oder weniger unverändert. Das Einzige, was anders ist, ist die Struktur der Branches innerhalb eines Repositorys. Statt eines einzelnen Master-Branchs verwendet der Gitflow-Workflow zwei Branches, um die Historie der Entwicklung abzubilden: zum einen den Master-Branch, welcher die offiziellen Releases enthält, zum anderen einen Entwicklungsbranch (in der Regel *dev* oder *develop* benannt), auf dem der aktuelle Entwicklungsstand enthalten ist.

Neue Features werden in neuen Branches entwickelt, allerdings nach Fertigstellung nicht direkt in den Master-Branch übertragen, sondern in den Entwickungsbranch. Darüber hinaus sieht der Gitflow-Workflow noch Release Bran-



Workflow-Variante: Der Forking-Workflow (Bild 14)



Workflow-Variante: Der Dictator-and-Lieutenants-Workflow (Bild 15)

ches vor, um gezielt an einzelnen Versionen einer Software arbeiten zu können, sowie Hotfix-Branches, um gezieltes Bugfixing zu betreiben.

Ein Workflow, der sich von den oben beschriebenen Workflows grundlegend unterscheidet, ist der Forking-Workflow beziehungsweise Integration-Manager-Workflow (Bild 14).

Statt eines zentralen serverseitigen Repositorys wie bei den anderen Workflows hat im Forking-Workflow jeder Entwickler ein eigenes serverseitiges Repository (Remote Repository). Eines dieser Repositories ist dabei das offizielle Repository, die anderen Repositories sind sogenannte Forks, also Kopien beziehungsweise Ableger von dem offiziellen Repository. Möchte nun ein Entwickler Änderungen, die er bereits in sein eigenes Remote Repository übertragen hat, auch in das offizielle Repository übertragen, muss er dazu einen Pull Request an den entsprechenden Entwickler des offiziellen Repositorys (den Integration Manager) stellen.

Bei der Arbeit mit vielen auf GitHub verfügbaren Open-Source-Bibliotheken ist der Forking-Workflow gang und gäbe (die Bibliothek jQuery beispielsweise hat momentan 9260 Forks). Möchte man eine Bibliothek nach seinen Wünschen anpassen oder ein neues Feature hinzufügen, erstellt man zunächst einen Fork der entsprechenden Bibliothek und arbeitet damit. Ist das Feature fertig, stellt man einen Pull Request an den Entwickler der Bibliothek.

Bei sehr komplexen Projekten, bei denen unterschiedliche Entwickler für bestimmte Teile eines Projekts beziehungsweise Subsysteme eines Projekts verantwortlich sind, kann das gerade beschriebene Konzept noch erweitert werden: Der sogenannte Dictator-and-Lieutenants-Workflow (Bild 15) sieht einen hierarchischen Aufbau vor, bei dem die für Subsysteme verantwortlichen Entwickler als Zwischenstufen (beziehungsweise als Lieutenants) fungieren, die den jeweiligen Code der Entwickler zunächst prüfen, bevor sie ihrerseits die Änderungen an eine als Chefentwickler bestimmte Person (den Dictator) weiterleiten. Nur dieser übernimmt die Änderungen dann in das offizielle Repository. Durch diesen mehrstufigen Prozess soll die Qualität des Codes sichergestellt werden.

Git Hooks

Ein Feature, das unter anderem auch für die Codequalität interessant ist, sind sogenannte Git Hooks. Dabei handelt es sich um Skripts, die ausgeführt werden, wenn ein bestimmtes Ereignis (wie beispielsweise das Übertragen von Daten in das lokale Repository oder das Übertragen von Daten in das Remote Repository) auftritt. Mit Hilfe solcher Skripts ist es möglich, verschiedenste Aspekte eines Workflows zu automatisieren oder zu optimieren.

Beispielsweise können Git Hooks dazu verwendet werden, vor jedem Commit die Unit-Tests für das entsprechende Projekt auszuführen und sicherzustellen, dass alle Tests erfolgreich abschließen. Für den Fall, dass ein Test fehlschlägt, könnte das Skript dann eine entsprechende Warnung ausgeben und verhindern, dass der Commit ausgeführt wird (so lange, bis die Tests wieder erfolgreich abschließen).

Git Hooks können in verschiedenen Programmiersprachen geschrieben werden, darunter Perl, Ruby oder Python. Auch für Node.js existieren verschiedene Module, die Git Hooks beispielsweise in Build-Tools wie Grunt oder Gulp integrieren. Prinzipiell ist man bei der Wahl der Programmiersprache relativ frei. Wichtig ist nur, dass sich das jeweilige Skript auf dem entsprechenden System ausführen lässt.

Git Hooks werden standardmäßig in dem Verzeichnis *hooks* unterhalb des *.git*-Verzeichnisses verwaltet. Dort sind nach Initialisierung eines Git-Repositorys bereits schon einige Beispielskripts enthalten. Um sie zu aktivieren, reicht es, das *sample* aus dem Dateinamen zu entfernen. ▶

Prinzipiell unterscheidet man zwischen clientseitigen und serverseitigen Hooks. Erstere werden beispielsweise im Rahmen von Commits oder im Rahmen vom Merging ausgeführt, Letztere beispielsweise beim Empfangen von Commits (sprich, wenn durch *git push* ein Commit an ein Remote Repository übertragen wurde). Eine Übersicht über die zur Verfügung stehenden Git Hooks zeigt Tabelle 3.

Der Hook *pre-commit* wird beispielsweise aufgerufen, bevor man eine Commit-Nachricht eingibt. Über diesen Hook hat man Gelegenheit, die Änderungen zu begutachten, die committet werden sollen, beispielsweise um wie erwähnt Unit-Tests auszuführen. Der Hook *commit-msg* wird ausgelöst, nachdem eine Commit-Nachricht eingegeben wurde. Dies kann hilfreich sein, um zu überprüfen, ob die Commit-Nachricht in einem bestimmten Format verfasst wurde, beispielsweise um sicherzustellen, dass sich aus den Commit-Nachrichten ein Change-Log generieren lässt (https://github.com/ajoslin/conventional-changelog).

Darüber hinaus stehen an clientseitigen Git Hooks noch weitere zur Verfügung: beispielsweise *post-merge*, welches ausgelöst wird, nachdem die Änderungen eines Branchs in einen anderen Branch übernommen wurden, oder *post-checkout*, nachdem von einem Branch in einen anderen Branch gewechselt wurde.

Bezüglich der serverseitigen Git Hooks stehen dagegen lediglich drei zur Verfügung: *pre-receive* und *update* werden ausgelöst, bevor mit dem Bearbeiten eines Push Requests begonnen wird. Der Unterschied: Der *update*-Hook wird für jeden Branch ausgelöst, der von Änderungen betroffen ist, der *pre-receive*-Hook dagegen nur einmal pro Push Request. Der *post-receive*-Hook wird nach dem Bearbeiten eines Push Requests ausgelöst.

Fazit

Hat man sich als Entwickler zu Zeiten von CVS und Subversion eher weniger mit dem Thema Versionskontrolle auseinandergesetzt, verlangt die Arbeit mit Git dem Entwickler viel mehr ab. Git unterscheidet sich dabei in vielerlei Hinsicht von anderen VCS. Zum einen handelt es sich bei Git um ein dezentrales VCS, zum anderen wird pro Commit ein kompletter Snapshot aller Dateien im Repository erzeugt. Darüber hinaus ist man bei der Zusammenstellung einzelner Commits dank des sogenannten Staging-Bereichs viel flexibler als bei anderen VCS. Über die Verwendung von Workflows lässt sich das Arbeiten im Team zudem weiter optimieren.

Neben den in diesem Artikel beschriebenen Workflows ist man dank der verschiedenen Konzepte, die Git zugrunde liegen, wie Branching, Forking und den dezentralen Repositories, relativ frei in der Definition eigener Workflows. ■

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.

Tabelle 3: Die verschiedenen Git Hooks

Git Hook	Wird ausgelöst	Typ
pre-commit	Vor dem Durchführen eines Commits	Clientseitig
prepare-commit-msg	Vor dem Verfassen einer Commit Message	Clientseitig
commit-msg	Nach dem Verfassen einer Commit Message	Clientseitig
post-commit	Nach dem Durchführen eines Commits	Clientseitig
applypatch-msg	Nach dem Verfassen einer Nachricht für einen Patch	Clientseitig
pre-applypatch	Vor dem Anwenden eines Patchs	Clientseitig
post-applypatch	Nach dem Anwenden eines Patchs	Clientseitig
pre-rebase	Vor dem Durchführen eines Rebase	Clientseitig
post-rewrite	Nach dem Ersetzen eines Commits, beispielsweise durch *git commit --amend* oder *git rebase*	Clientseitig
post-checkout	Nach dem Wechseln eines Branchs (beziehungsweise nach dem Ausführen von *git checkout)*	Clientseitig
post-merge	Nach dem Merging	Clientseitig
pre-push	Vor dem Übertragen von Daten an das Remote Repository	Clientseitig
pre-auto-gc	Vor dem Ausführen der Garbage Collection	Clientseitig
pre-receive	Vor dem Empfangen eines Push Requests	Serverseitig
update	Ähnlich wie *pre-receive,* nur dass es für jeden Branch aufgerufen wird, der von Änderungen betroffen ist	Serverseitig
post-receive	Nach dem Empfangen und Bearbeiten eines Push Requests	Serverseitig

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

SCHNELLER CSS ERZEUGEN MIT POSTCSS

Statt Sass und LESS

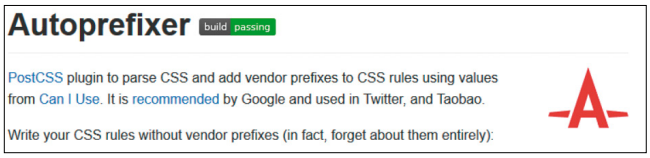
Das neue Tool PostCSS will die etablierten Präprozessoren ablösen.

CSS ist im Grunde einfach: Elemente auswählen und Regeln anwenden. Dass CSS in der Praxis aber nicht so einfach ist, liegt an der schieren Masse an Regeln, die die Stylesheets gängiger Projekte brauchen. Ein paar Tausend Zeilen sind da keine Ausnahme, wobei viele Angaben mehr oder weniger stark variierte Wiederholungen sind – und das macht Wartungs- und Änderungsarbeiten aufwendig.

Das andere Problem sind die Browser: Zwar haben wir nicht mehr die massiven Browserunterschiede samt all den notwendigen Browserhacks wie zu Zeiten der Browserkriege; dafür werden aber mit CSS3 unglaublich viele neue Regeln und Eigenschaften eingeführt, die von den einzelnen Browsern unterschiedlich unterstützt werden.

Erschwerend kommt hinzu, dass CSS keine Programmiersprache ist und deswegen nicht die Features bietet, die jede Feld-Wald-und-Wiesen-Programmiersprache mit sich bringt, etwa Variablen, Schleifen oder Konditionen. Letzteres ändert sich jedoch durch die in CSS3 vorgesehenen Variablen, Media Queries (die ja auch so etwas sind wie Bedingungen) oder *@supports* zur Abfrage der Unterstützung.

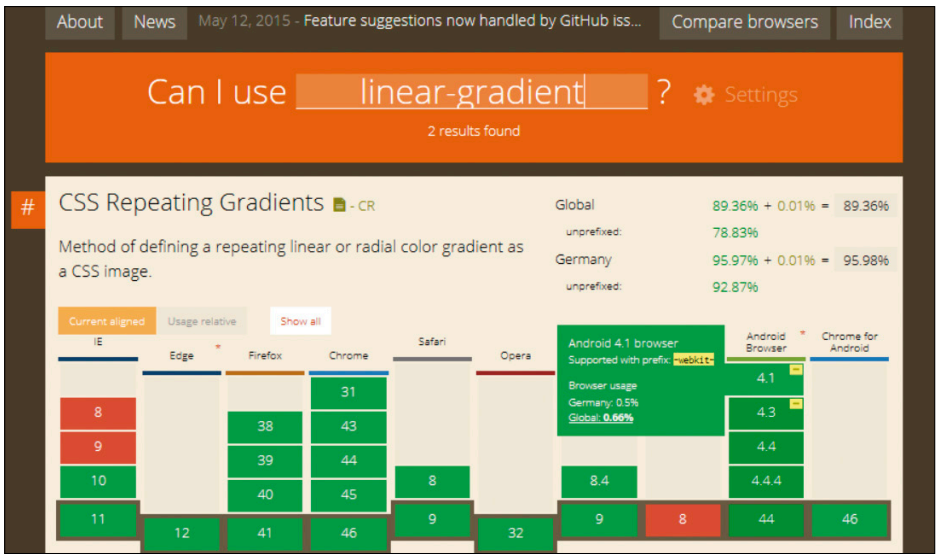
Kein Wunder, dass es Präprozessoren gibt, die da nachbessern und bei der Erstellung von Stylesheets helfen. Am bekanntesten sind wohl Sass, LESS und Stylus. Auch wenn sich die Tools im Detail unterscheiden, ist ihre grundlegende Arbeitsweise doch gleich; im Folgenden konzentrieren wir uns beispielhaft auf Sass. Wenn Sie einen CSS-Präprozessor verwenden, schreiben Sie keinen normalen CSS-Code, sondern CSS-Code mit Sass-Direktiven. Das ist ähnlich wie die Template-Systeme, die man beispielsweise auch verwendet, um über PHP Inhalte in HTML-Seiten zu injizieren. Dieser Code wird dann kompiliert und daraus das Stylesheet erzeugt. Im Optimalfall ist der Sass-Code wesentlich übersichtlicher und kürzer als das generierte Stylesheet und damit schneller zu erstellen und besser zu warten. Kein Wunder, dass CSS-Präprozessoren so beliebt sind.



Autoprefixer ist eines der bekanntesten PostCSS-Plug-ins (Bild 1)

Allerdings hat dieser Ansatz auch Nachteile:

- Komplexe Mixins in Sass zu schreiben erfordert Einarbeitungszeit. Das dadurch gewonnene Wissen können Sie außerhalb der Sass-Welt nicht weiterverwenden.
- Sass ist ein großer, monolithischer Block: Wenn Sie Sass einsetzen, haben Sie die gesamten Möglichkeiten von Sass. Es gibt jedoch keinen vorgegebenen Weg, um bestimmte Features nicht zu erlauben, also beispielsweise *@extend* nicht zu erlauben. Modularität erhalten Sie nur durch Mixin-Bibliotheken wie Compass.
- Wenn ein gewünschtes Feature erst in der nächsten Version verfügbar ist, kann man nur abwarten.
- Die Erzeugung von browserspezifischem Code – zum Beispiel die Ergänzung von Eigenschaften mit herstellerspezifischen Präfixen – ist nicht sehr elegant.



Autoprefixer verwendet das Datenmaterial von Can I use (Bild 2)

Heute sind wesentlich seltener herstellerspezifische Präfixe wie *-webkit-*, *-moz-* und *-ms-* notwendig als noch vor ein paar Jahren. Das liegt daran, dass Browserhersteller neue Eigenschaften üblicherweise nicht mehr zuerst mit einem Präfix testweise implementieren, sondern hinter einem Flag, das interessierte Webentwickler aktivieren können.

Trotzdem existieren noch Fälle, in denen Sie herstellerspezifische Präfixe brauchen. Zudem gibt es Situationen, in denen Sie unter-

schiedliche Eigenschaften für verschiedene Browser benötigen – beispielsweise bei Flexbox. In der Sass-Welt können Sie für die Präfixerzeugung Compass nutzen. Compass sieht dafür eigene Mixins vor. Wenn Sie bei Transformationen die richtigen Präfixe ergänzen lassen wollen, so schreiben Sie:

```
.beispiel {
  @include transform(translateX(2em));
}
```

Inzwischen ergänzt Compass nicht einfach nur wahllos alle möglichen Präfixe, sondern zieht zur Ermittlung dessen, was notwendig ist, die Angaben von *Can i use* heran. Sie können konfigurieren, welche Browserversionen Sie berücksichtigen möchten.

Speziell zur Erzeugung browserspezifischen Codes ist in letzter Zeit Autoprefixer (Bild 1) attraktiv geworden. Autoprefixer greift ebenfalls auf die Daten von *Can i use* (Bild 2) zurück und erzeugt die benötigten Präfixe der von Ihnen gewählten Browser. Statt dafür aber Präprozessor-spezifischen Code, das heißt, den Aufruf eines Mixins, schreiben zu müssen, verwenden Sie direkt den Standard-CSS-Code – um den Rest kümmert sich Autoprefixer automatisch:

```
.beispiel {
  transform: translateX(2em);
}
```

Das erzeugte Stylesheet sieht – sofern dieselben Browser bei der Konfiguration ausgewählt wurden – bei der Verwendung von Compass oder Autoprefixer identisch aus – ergänzt werden die Präfixe für ältere WebKit-Browser und für den Internet Explorer 9:

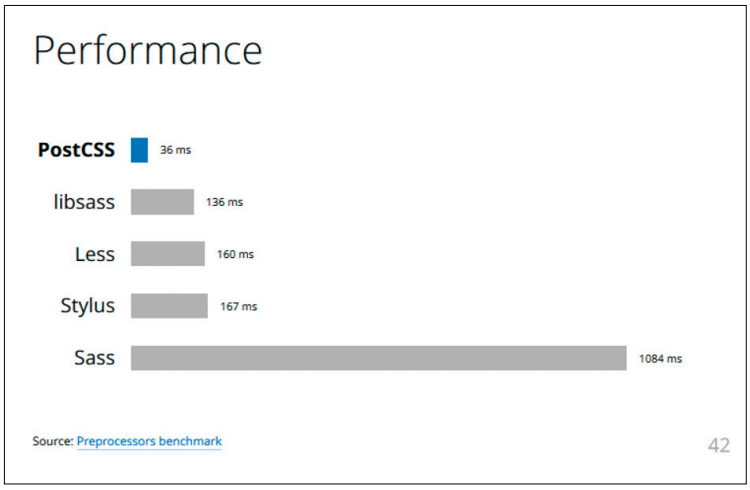
```
.beispiel {
  -webkit-transform: translateX(2em);
  -ms-transform: translateX(2em);
  transform: translateX(4em);
}
```

Autoprefixer ist das Tool, das PostCSS populär gemacht hat. Sehen wir uns jetzt einmal das Grundprinzip von PostCSS an.

Grundprinzip von PostCSS

Die Gemeinsamkeit von CSS-Präprozessoren und PostCSS (Bild 3) ist das Ziel, effektiver und schneller CSS-Code zu schreiben. In der Vorgehensweise unterscheiden sich die beiden jedoch.

PostCSS selbst ist eigentlich nur ein Parser, ein API für node trees, ein SourceMap-Generator und ein Stringifier. Kurz gesagt stellt PostCSS alles für die Bearbeitung und Modifizierung von Stylesheets zur Verfügung. Die eigentliche Bearbeitung findet aber nicht in PostCSS selbst statt, sondern über die PostCSS-Plug-ins. Wenn Sie PostCSS ohne Plug-in laufen lassen, gibt es keinen Unterschied zwischen Input- und Output-Stylesheet. Damit verfolgt PostCSS einen modularen Ansatz: Sie entscheiden, welche Modifikationen Sie durchführen möchten, und binden die entsprechenden Plug-ins ein.

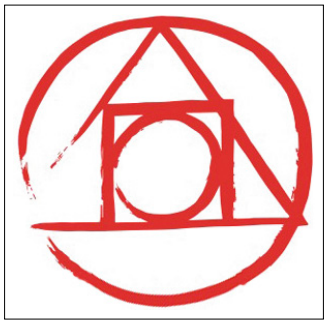


Vergleich der Performance von Präprozessoren und PostCSS (Bild 4)

Es gibt PostCSS-Plug-ins für die unterschiedlichsten Zwecke. Neben dem erwähnten Autoprefixer zur Ergänzung der Browserpräfixe und der von einzelnen Browsern benötigten Sonderangaben wäre da beispielsweise *cssnext*, mit dem Sie CSS-Features der nächsten Generation heute schon nutzen können, auch wenn sie noch nicht in den Browsern angekommen sind.

Variablen oder Mixins

Aber auch die andere Richtung kann sinnvoll sein: Falls Sie einmal CSS-Angaben für Uralt-IEs benötigen, so kümmert sich das PostCSS-Plug-in *cssgrace* darum. Außerdem gibt es noch Plug-ins für Features, die Sie von Sass kennen, wie Variablen oder Mixins, und Sie finden Plug-ins, um den Code zu optimieren oder um Sie bei bestimmten Angaben zu warnen.



Das Logo von PostCSS (Bild 3)

Dem monolithischen Block eines CSS-Präprozessors mit vordefinierten Features steht also bei Post-CSS ein Plug-in-System gegenüber. So können Sie je nach Projekt unterschiedliche Plug-ins nutzen. Falls eine gewünschte Modifikation nicht vorhanden sein sollte, können Sie auch Ihr eigenes Plug-in schreiben. Dafür müssen Sie keine eigene Sprache lernen, sondern Sie setzen JavaScript ein. Dass die Hürde zur Erstellung von Plug-ins so gering ist, begünstigt neue Plug-ins – so wird die Auswahl immer größer.

Die prinzipielle Herangehensweise bei CSS-Präprozessoren und PostCSS unterscheidet sich noch in einem weiteren Punkt: Während Sass und Co. Templates sind – Styles und Logik sind in einer Datei kombiniert –, ist der Input bei PostCSS meist reines CSS, die Logik steckt in den Plug-ins. Ein weiterer Vorteil ist schließlich, dass PostCSS wesentlich schneller ist als Sass oder LESS (Bild 4). ▶

PostCSS können Sie mit allen klassischen Build-Tools verwenden – natürlich mit Grunt oder Gulp (Bild 5), aber auch mit webpack, Broccoli, Brunch, ENB, Fly, Stylus, Duo und Connect/Express. Außerdem gibt es noch postcss-js, das Sie in React Inline Styles, Free Style oder Radium nutzen können, und es existiert ein Kommandozeilen-Tool. Sehen wir uns am Beispiel von Gulp an, wie die Arbeit mit PostCSS funktioniert.

PostCSS mit Gulp

Für Gulp benötigen Sie Node.js, das Sie vorher herunterladen müssen. Ein Aufruf von *node –v* zeigt, ob es geklappt hat und welche Version installiert ist. Als Nächstes installieren Sie Gulp:

```
npm install gulp -g
```

Zu Überprüfung, ob alles geklappt hat, dient wiederum:

```
gulp -v
```

Wechseln Sie dann in das Verzeichnis Ihres Projekts mit:

```
cd pfadzumprojekt
```

Dort installieren Sie Gulp lokal über:

```
npm install gulp --save-dev
```

Und schließlich müssen Sie noch PostCSS installieren:

```
npm install --save-dev gulp-postcss
```

Um besser testen zu können, ob alles geklappt hat, sollten Sie zusätzlich ein Plug-in installieren, beispielsweise Autoprefixer:

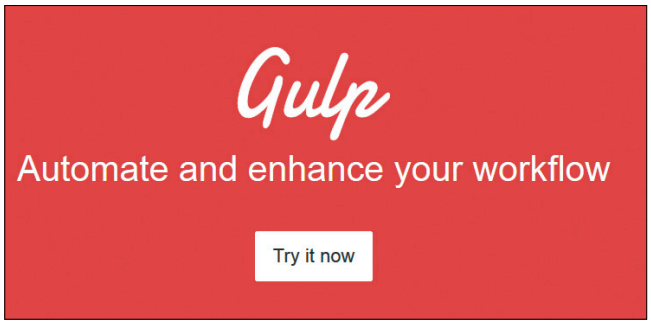
```
npm install --save-dev autoprefixer
```

Jetzt fehlt noch das *gulpfile.js* für die Konfiguration, das sich im Projektverzeichnis befinden muss. In diesem legen wir Variablen für die Komponenten an:

```
var postcss = require('gulp-postcss');
var gulp = require('gulp');
var autoprefixer = require('autoprefixer');
```

Danach definieren wir einen Task mit dem Namen *css*, in dem wir Autoprefixer als Prozessor angeben und konfigurieren: Es sollen die letzten drei Browserversionen berücksichtigt werden und alle Browser mit mehr als einem Prozent Nutzer:

```
gulp.task('css', function () {
  var processors = [
    autoprefixer({browsers: ['last 3 versions', '> 1%']})
  ];
  return gulp.src('./src/*.css')
    .pipe(postcss(processors))
    .pipe(gulp.dest('./dest'));
});
```



PostCSS funktioniert mit Gulp, aber auch mit anderen Build-Tools wie Grunt (Bild 5)

Zusätzlich ist angegeben, dass sich die Dateien, die verarbeitet werden sollen, im Verzeichnis *src* des aktuellen Projektordners befinden und die Endung *css* haben. Das Ergebnis soll in einem Ordner *dest* gespeichert werden, der bei Bedarf automatisch angelegt wird.

Führen wir einen Test durch. Im Ordner *src* befindet sich die Datei *style.css* mit folgendem Inhalt:

```
.a {
  display: flex;
  background: linear-gradient(red, green);
}
```

Starten wir den Task durch die Eingabe des Befehls:

```
gulp css
```

Danach finden wir im Ordner *dest* die neue Datei *style.css*. Wie zu erwarten, wurden die browserspezifischen Angaben bei Flexbox und für den Farbverlauf ergänzt:

```
.a {
  display: -webkit-box;
  display: -webkit-flex;
  display: -ms-flexbox;
  display: flex;
  background: -webkit-linear-gradient(red, green);
  background: linear-gradient(red, green);
}
```

Was etwas irritierend ist: Tools wie Autoprefixer gibt es in verschiedenen Varianten, auch als eigenständiges Gulp-Plug-in, das dann den Namen *gulp-autoprefixer* hat. Aber das funktioniert dann standalone, während wir es hier ja innerhalb von PostCSS einsetzen wollen.

Mehrere Plug-ins nutzen

Im Normalfall werden Sie mehrere Plug-ins nutzen. Im Beispiel soll zusätzlich zu Autoprefixer *cssnext* zum Einsatz kommen. Zuerst müssen Sie das Plug-in installieren:

```
npm install --save-dev postcss-cssnext
```

Dann geht es an die Anpassung der Datei *gulpfile.js*. Dort definieren wir eine zusätzliche Variable *cssnext* für das zusätzliche Plug-in.

```
var postcss = require('gulp-postcss');
var gulp = require('gulp');
var autoprefixer = require('autoprefixer');
var cssnext = require('postcss-cssnext');
```

Außerdem führen wir *cssnext()* bei den Prozessoren auf:

```
gulp.task('css', function () {
  var processors = [
    autoprefixer({browsers: ['last 3 versions',
    '> 1%']}), cssnext()
  ];

  return gulp.src('./src/*.css')
  .pipe(postcss(processors))
  .pipe(gulp.dest('./dest'));
});
```

Jetzt können wir in unserem Stylesheet auch selbstdefinierte Variablen einsetzen – im Beispiel definieren wir eine Variable *--heading*, die alle Überschriften *h1 – h6* auswählt. Für diese legen wir einen oberen Abstand von *0* fest.

```
@custom-selector :--heading h1, h2, h3, h4, h5, h6;
:--heading { margin-top: 0 }
```

Dank *cssnext* werden diese Angaben in Code übersetzt, den die heutigen Browser verstehen:

```
h1, h2, h3, h4, h5, h6 { margin-top: 0 }
```

Sehen wir uns die Konfigurationsmöglichkeiten des beliebtesten PostCSS-Plug-ins – Autoprefixer – im Folgenden einmal genauer an.

Autoprefixer im Detail

Welche Browser berücksichtigt werden sollen, können Sie auf verschiedene Arten angeben. In den Beispielen bisher wurden zwei Varianten eingesetzt:

- Variante 1: *last X versions* bedeutet, dass nur die X letzten Versionen berücksichtigt werden.
- Variante 2: *> Y%* bestimmt, dass nur der Code für Browser, die mehr als Y% Nutzer haben, geschrieben werden soll.

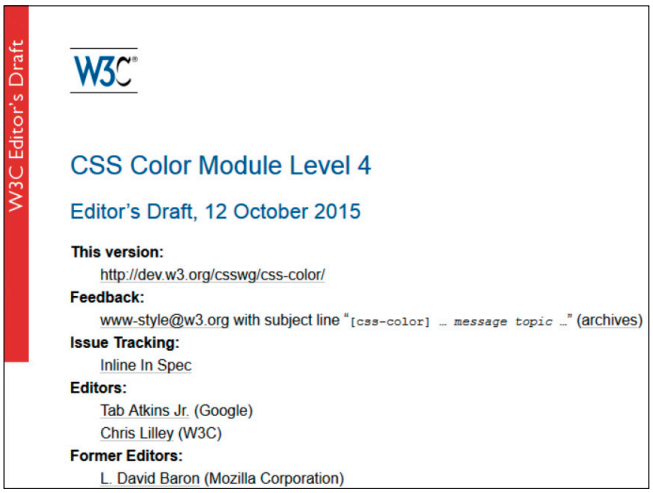
Zusätzlich können Sie einzelne Browser mit Versionen angeben wie *Firefox >=20* für Firefox ab Version 20; *ie 6-8* wählt entsprechend den Internet Explorer 6, 7 und 8 aus. Sie können dabei folgenden Browsernamen – bei manchen gibt es auch Abkürzungen – einsetzen: Android, BlackBerry (bb), Chrome, Firefox (ff), Explorer (ie), Edge, iOS (ios_saf), Opera, Safari, OperaMobile (op_mob), OperaMini (op_mini), ChromeAndroid (and_chr), FirefoxAndroid (and_ff) oder ExplorerMobile (ie_mob).

Autoprefixer verwendet dabei Browserlist zur Angabe, welche Browser berücksichtigt werden sollen. Angaben nach Version und Nutzung können Sie bei Bedarf auch kombinieren; dabei werden die einzelnen Angaben jeweils in Anführungszeichen angegeben und durch ein Komma getrennt: *'last 3 versions', '> 5%'*.

Veraltete Angaben entfernen

Autoprefixer ergänzt übrigens nicht nur zusätzliche Eigenschaften mit Präfixen, sondern entfernt darüber hinaus veraltete Angaben wie *-moz-border-radius* ebenso wie gänzlich sinnlose Einträge wie *-ms-border-radius* (da der Internet Explorer die Eigenschaft *border-radius* direkt implementiert hat, gab es nie einen Browser, der die Angabe *-ms-border-radius* interpretiert hat).

Falls Sie in einem speziellen Fall die Entfernung alter Präfixe nicht wünschen, so können Sie bei der Konfiguration von Autoprefixer *remove: false* angeben:



Mit cssnext können Sie heute schon die neuen Farbangaben aus dem CSS Color Module Level 4 nutzen (Bild 6)

```
var processors = [
  autoprefixer({browsers: ['last 3 versions', '> 1%'],
  remove: false})
];
```

Weitere Optionen sind *add: false*, sofern keine Präfixe ergänzt werden sollen, und *cascade*, über das Sie steuern können, ob bei einem nicht minimierten Stylesheet zugehörige Selektoren eingerückt werden sollen (Standard: *true)*.

Back to the future

Das CSS Color Module Level 4 des W3C (Bild 6) sieht ein verbessertes Handling bei Farbangaben vor. Zuerst einmal werden Lücken gefüllt. Warum kann man eigentlich bei einer hexadezimalen Farbangabe keine Transparenz festlegen? *hexa* macht genau das möglich und verhält sich zu einer ▶

Cssgrace kümmert sich um alte IEs, die gerade in China noch wichtig sind – die Erklärungen gibt es aber auch auf Englisch (Bild 7)

klassischen Hexadezimal-Angabe wie *rgba()* zu *rgb()*. Bei *hexa* geben Sie eine weitere zweistellige Hexadezimalzahl am Ende an, die den Grad der Transparenz festlegt. *#ff000080* definiert zum Beispiel ein halb transparentes Rot.

Einleuchtend ist auch die neue Funktion *gray()*, mit der Sie einen Grauton festlegen können, ohne dass Sie einen Farbwert angeben müssen, der ja an sich bei Grautönen sinnlos ist. Nützlich ist ebenfalls die *color()*-Funktion für Farbanpassungen.

W3C-Spezifikationen

Aber die nächsten Levels der W3C-Spezifikationen haben noch mehr zu bieten – so beispielsweise selbst definierte Selektoren. Die Browserunterstützung dieser neuen Features ist jedoch mager bis gar nicht vorhanden. Trotzdem können Sie diese Angaben dank *cssnext* heute schon schreiben. *cssnext* verwandelt diese neue Syntax in Angaben, die von den heutigen Browsern verstanden werden.

Setzen wir die neuen Farbangaben einmal ein: Zuerst verwenden wir *hexa* für den Hintergrund, die *color()*-Funktion für den Rahmen (im Beispiel wird zu Rot eine weitere Farbe addiert) und schließlich legen wir über *gray()* den Grauton fest – der erste Parameter bei *gray()* bestimmt die Helligkeit, der zweite die Transparenz:

```
.b {
  background-color: #11234566;
  border: 1px solid color(red rgb(+ #004400));
  color: gray(10%, 60%);
}
```

Nach der Bearbeitung durch *cssnext* wurden im Ergebnis-Stylesheet aus den neuen Farbangaben *rgba()*-Angaben erzeugt, zusätzlich sind Fallback-Farben ohne Alphakanal für ältere Browser ergänzt:

```
.b {
  background-color: #112345;
  background-color: rgba(17, 35, 69, 0.4);
  border: 1px solid rgb(255, 0, 0);
  color: #1A1A1A;
  color: rgba(26, 26, 26, 0.6);
}
```

Schön ist auch die Unterstützung für CSS3-Filter von *cssnext*, dabei wird der SVG-Code für ältere Firefox-Browser erzeugt.

Mehr zur Browserunterstützung

Während *cssnext* in die Zukunft gerichtet ist, kümmert sich das Gegenstück *cssgrace* um die Vergangenheit (Bild 7) – es generiert, wo möglich, den für ältere Browser benötigten Code, also beispielsweise für Internet Explorer 6, 7 oder 8.

Das ist für bestimmte Länder wie China relevant, weil dort der Anteil dieser älteren IEs wesentlich höher ist. *cssgrace* erzeugt dann Hacks wie die proprietären Microsoft-Filter-Angaben automatisch. Aus:

```
.foo {
  opacity: .6;
}
```

… wird etwa:

```
.foo {
  opacity: .6;
  filter: alpha(opacity=60);
}
```

Ein weiteres Tool zum Umgang mit Browserunterschieden ist *doiuse*. Es setzt wie Autoprefixer auf die Daten von *Can I use*. Allerdings ergänzt *doiuse* keinerlei Code, sondern gibt Ihnen Informationen über Features aus, die von bestimmten Browsern nicht unterstützt werden. Die Browser, die dabei berücksichtigt werden sollen, können Sie gesondert angeben.

Mehr Plug-ins

Auch wenn Sie auf PostCSS setzen, müssen Sie Ihre Sass-Gewohnheiten nicht über Bord werfen. Wenn Sie sich an Mixins gewöhnt haben, können Sie sie auch mit PostCSS weiternutzen – dafür gibt es das Plug-in *postcss-mixins,* während *postcss-nested* es erlaubt, Regeln in Sass-Manier zu verschachteln, und *postcss-simple-vars* Variablen einführt, wie sie von Sass bekannt sind.

Eine Alternative dazu ist *precss,* das gleich mehrere Sass-Features auf einmal bietet. Allerdings sollten Sie bei Plug-ins, die eine andere Syntax erlauben, als in normalem CSS üblich ist, diese explizit in Ihrem Basis-Stylesheet ergänzen. Dafür können Sie auf *postcss-use* zurückgreifen und die Plug-ins über *@use* zu Beginn des Stylesheets definieren.

Ein neuer Trend sind Quantity-Queries. Diese bieten die Möglichkeit, unterschiedliche Formatierungen bereitzustellen, je nachdem, wie viele Elemente es in einem Kontext gibt. So könnte man beispielsweise bei vier Elementen die Breite auf 25 Prozent setzen wollen, bei drei Elementen auf 33 Prozent et cetera. Wie viele Elemente es aber sind, weiß man nicht im Voraus – eine nicht untypische Situation beim Einsatz von Content-Management-Systemen. Diese quantitati-

Links zum Thema

- Compass
  http://compass-style.org
- PostCSS
  https://github.com/postcss/postcss
- Node.js
  https://nodejs.org/en
- Browserlist
  https://github.com/ai/browserslist
- Color-Module Level 4
  https://drafts.csswg.org/css-color
- Katalog von PostCSS-Plug-ins
  http://postcss.parts
- Im Artikel erwähnte PostCSS-Plug-ins
  https://github.com/postcss/autoprefixer
  https://github.com/cssdream/cssgrace
  https://github.com/anandthakker/doiuse
  https://github.com/postcss/postcss-mixins
  https://github.com/postcss/postcss-nested
  https://github.com/postcss/postcss-simple-vars
  https://github.com/jonathantneal/precss
  https://github.com/postcss/postcss-use
  https://github.com/pascalduez/postcss-quantity-queries

ven Abfragen können Sie durch fortgeschrittene, aber recht komplexe Selektoren umsetzen. Dank *postcss-quantity-queries* müssen Sie sich nun um die Umsetzung nicht mehr kümmern. Wenn etwa eine Formatierung nur gelten soll, wenn es vier bis sechs Elemente sind, schreiben Sie:

```
ul > li:between(4, 6) {
  color: rebeccapurple;
}
```

postcss-quantity-queries erzeugt Ihnen dann die dafür benötigten komplexen Selektoren:

```
ul > li:nth-last-child(n+4):nth-last-child(-n+6):
first-child,

ul > li:nth-last-child(n+4):nth-last-child(-n+6):
first-child ~ li {
  color: rebeccapurple;
}
```

Aber natürlich gibt es noch viele weitere interessante Plug-ins, wie zum Beispiel *stylelint*, das den CSS-Code verbessert, *epub*, das das *epub*-Präfix für E-Books ergänzt, *will-change*, das den notwendigen 3D-Hack vor dieser neuen Eigenschaft hinzufügt, *pixrem* für Fallback-Angaben in Pixeln, *vertical-rhythm* für einen vertikalen Rhythmus, *colorblind*, das die Farben manipuliert, um verschiedene Formen von Farbenblindheit zu simulieren; *cssbyebye* entfernt ausgewählte Regeln, *focus* ergänzt zu jedem *:hover*-Selektor auch den *:focus*.

Diese kleine Auswahl soll die Vielfältigkeit der Plug-ins demonstrieren – ein Blick in den Plug-in-Katalog mit dem gesamten Verzeichnis lohnt sich. Und falls Sie dort nicht das Gewünschte finden, können Sie eigene Plug-ins in JavaScript schreiben. Das Schöne daran ist, dass einfache Plug-ins auch nur ein paar Zeilen Code beinhalten.

PostCSS und Präprozessoren

Es ist natürlich eine Vereinfachung zu behaupten, dass PostCSS die CSS-Präprozessoren ersetzt; das Verhältnis von PostCSS zu CSS-Präprozessoren ist vielfältiger. Zum einen können Sie selbstverständlich CSS-Präprozessoren und PostCSS gleichzeitig nutzen: Sinnvoll wäre es beispielsweise, den Code in Sass zu schreiben, zu kompilieren und diese CSS-Datei als Input für PostCSS zu nutzen, um weitere Operationen nach Bedarf durchführen zu lassen – beispielsweise an dieser Stelle Autoprefixer zu nutzen.

Zum anderen können Sie Sass-Komponenten, an die Sie sich gewöhnt haben, auch über die erwähnten PostCSS-Plug-ins nutzen wie precss. Allerdings dürfen Sie dabei nicht erwarten, dass eine hundertprozentige Übereinstimmung besteht, und Sie sollten nicht ohne detaillierte Tests bei einem Sass-Projekt rein auf PostCSS umsteigen.

Fazit

Modulare Herangehensweise, eine Fülle von Plug-ins und performant – das sind wichtige Merkmale von PostCSS. Ein weiterer Vorteil ist, dass populäre Plug-ins auch die Weiterentwicklung von CSS an sich beeinflussen können – so wie wir es bei jQuery und JavaScript sehen.

Natürlich hat PostCSS auch Nachteile: Wenn Sie einen neuen Entwickler ins Team holen und sagen, dass Sie Sass benutzen, ist die Sache (halbwegs) klar. Bei PostCSS müssen Sie hingegen angeben, welche Plug-ins Sie nutzen. Denn je nach benutzten Plug-ins wird der geschriebene CSS-Code ganz unterschiedlich aussehen.

Wichtig ist außerdem – wie meistens –, dass Sie wissen, was Sie tun. Durch die Verwendung von *cssnext* könnte man beispielsweise leicht aus dem Blick verlieren, dass der geschriebene Code heute noch nicht in den Browsern funktioniert. Ähnlich könnte es auch mit Autoprefixer sein. Wer sich angewöhnt, *.beispiel { column-count: 3 }* zu schreiben, vergisst vielleicht, dass er bei einem CSS-pur-Projekt noch die anderen Angaben ergänzen muss. ■

Dr. Florence Maurice
ist Autorin, Trainerin und Programmiererin in München. Ihr aktuelles Buch ist dem Thema »CSS3« gewidmet und bei dpunkt erschienen (ISBN 978-3-86490-118-8). Außerdem bloggt sie regelmäßig zu CSS und verwandten Themen unter: **http://maurice-web.de/blog**

WEB FRONTEND DEVELOPMENT MIT ANGULAR 2.0

Template-Syntax

Das Templating mit AngularJS war bereits ein mächtiges Werkzeug. Mit Angular 2.0 legen die Entwickler nun kräftig nach.

Der vorangegangene Artikel »Angular 2.0 und modularer Code« erläuterte, wie mit SystemJS Bibliotheken und eigene Client-Side-Skripts geladen und ausgeführt werden können. Ein erstes Hello-World-Beispiel mit Angular 2.0 wurde entwickelt. Das heißt, dass die Ausführung von ECMA-Script-6-Modulen nun keine Hürde mehr ist. Es wird Zeit, tiefer in das Framework einzutauchen.

Dieser Artikel stellt die neue Template-Syntax von Angular 2.0 vor. Es halten zahlreiche, neue Möglichkeiten Einzug, um Oberflächenelemente zu beschreiben. Die Entwickler von Angular verfolgen hierbei ein großes Ziel: das Konzept der Template-Syntax eindeutiger und nachvollziehbarer zu formulieren, als es bei der Vorgängerversion der Fall ist. Auch der Support durch Editoren, etwa durch bessere automatische Vervollständigung, soll nun verbessert werden und die Produktivität des Entwicklers steigern.

Zur näheren Erläuterung wird ein Prototyp genutzt, der als Dashboard für Schäden an Autos dienen soll (Bild 1).

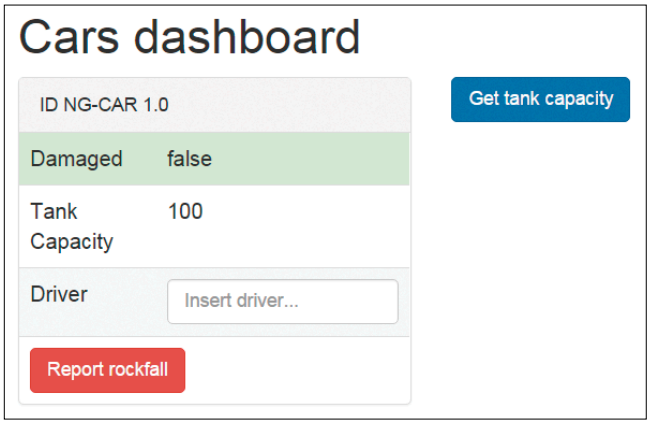
Neben einer ID und dem Schadensstatus kann auch der aktuelle Füllstand des Fahrzeugs abgefragt werden. Des Weiteren kann mit einem Klick ein Steinschlag (rockfall) gemeldet werden.

Übrigens Sie finden das hier vorgestellte Beispiel auf GitHub unter http://bit.ly/web_und_mobile_angular2_artikel2 und auf www.webundmobile.de.

Components und Views

Angular-2-Anwendungen bestehen aus verschiedenen Komponenten (Components), die miteinander agieren können. Für das Dashboard wird eine Komponente benötigt. Im Dashboard wird eine Liste von Autos abgebildet werden. Hierfür wird eine weitere Komponente implementiert. Den Aufbau einer Angular-2.0-Komponente zeigt Listing 1.

Von Angular werden zunächst zwei Module *@Component()* und *@View()* importiert. Diese beiden Module sind im Speziellen TypeScript-Dekoratoren. Dekoratoren ermöglichen es, Klassen durch Meta-Angaben erweitern. *@Component()* spezifiziert, dass die Dashboard-Komponente über den Selector *<dashboard>* im DOM des HTML-Dokuments eingesetzt wird. Mit *@View()* definiert man das Template, das mit der Komponenten verknüpft ist. In diesem Beispiel wird das Feld *id* aus der Klasse *DashboardComponent* im Template gebunden und angezeigt. An dieser Stelle wird deutlich, was eine Komponente ist: Komponenten sind die neuen zentralen Bausteine von Angular 2.0. Sie übernehmen die Rolle von Direktiven und Controllern aus AngularJS.



Dashboard-Prototyp: Dashboard für die Eingabe von Schäden an Autos (Bild 1)

Eine Komponente ist ein angereichertes Template, das im Browser zur Anzeige gebracht wird. Das Template wiederum verfügt über ein spezifisches Verhalten, das in Angular 2.0 durch TypeScript-Dekoratoren beschrieben wird.

Interpolation

Wie wird nun aus dem Ausdruck *{{ id }}* ein angezeigter Text im Browser?

Bereits in AngularJS 1.x konnten Daten mit Hilfe zweier geschweifter Klammern an ein HTML-Template gebunden werden. Der Wert wurde dann mittels Interpolation ausgewertet und angezeigt. Dieses Konzept bleibt in Angular 2.0 erhalten:

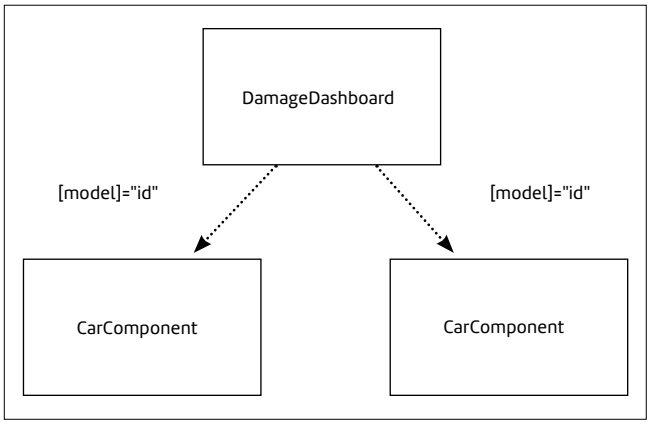
```
<p>{{ id }}</p>
```

Listing 1: Aufbau eine Komponente in Angular 2.0

```
import { Component, View } from 'angular2/angular2';

@Component({ selector: 'dashboard' })
@View({
  template: `<p>{{ id }}</p>`
})
export default class DashboardComponent {
  id: string = 'NG-Car 2015';
}
```

Listing 2: CarComponent

```
import { Component, View, Input } from
'angular2/angular2';

@Component({ selector: 'car' })
@View({
  template: `<p>{{ id }}</p>`
})
export default class CarComponent {
  @Input() id: string;
}
```



Daten an eine Komponente übergeben (Property-Binding) (Bild 2)

Die Schreibweise ist im Übrigen eine Vereinfachung der tatsächlichen Syntax. Denn bevor dieses Template im Browser ausgegeben wird, setzt Angular den Ausdruck in ein Property-Binding um:

```
<p [text-content]="interpolate(['Gregor'], [name])"></p>
```

Um in dem Dashboard nun ein Auto abbilden zu können, wird eine weitere Komponente benötigt (Listing 2).

Im ersten Schritt soll diese Komponente lediglich die zugewiesene Identifikationsnummer ausgeben. Der *@Input()*-Dekorator bietet die Möglichkeit, Werte an die *CarComponent* zu übergeben. Nun kann die *CarComponent* im Dashboard referenziert und im Template verwendet werden (Listing 3).

Im Wesentlichen wurden drei Anpassungen vorgenommen:

- Über ein weiteres *import*-Statement wird *CarComponent* geladen.
- *@View()* wird durch die Eigenschaft *directives* ergänzt, damit *CarComponent* im Template verwendet werden kann.
- Das Feld *id* wird an die gleichnamige Eigenschaft der *CarComponent* gebunden. Hierbei handelt es sich um ein Property-Binding.

So wurde über die Datenbindung die erste Interaktion zwischen zwei Komponenten realisiert.

Listing 3: dashboard.component.ts

```
import { Component, View } from 'angular2/angular2';
import { CarComponent } from '../car/car.component';

@Component({ selector: 'dashboard' })
@View({
  directives: [CarComponent],
  template: `<car [id]="id"></car>`
})
export default class DashboardComponent {
  id: string = 'NG-Car 2015';
}
```

Input- und Output-Properties sind Eigenschaften, die das API einer Angular-Komponente beschreiben. Über Inputs werden Informationen an eine Komponente übergeben. Mit Outputs kommuniziert die Komponente Änderungen nach außen. Inputs werden durch Property-Bindings beschrieben. Outputs können über Event-Bindings abonniert werden. Mit Properties werden einer Komponente Daten übermittelt (Bild 2). Property-Bindings zeichnen sich durch eckige Klammern aus (*[id]*):

```
<car [id]="id"></car>
```

Anstatt eckiger Klammern können Property-Bindings auch mit der vollständigen Syntax *bind-{property-name}="{expression}"* beschrieben werden:

```
<car bind-id="id"></car>
```

Events bieten die Möglichkeit, auf Veränderungen einer Komponente zu reagieren. Anders gesagt bieten sie einer Komponente also die Möglichkeit, mit ihrer Außenwelt zu kommunizieren (Bild 3).

Event-Bindings zeichnen sich durch runde Klammern aus. Sie triggern die Ausführung eines Statements:

```
<car (damaged)="report(damage)"></car>
```

Auch für diese Syntax existiert eine längere Syntax in der Form *on-{event-name}="{statement}"*:

```
<car on-damaged="report(damage)"></car>
```

Um ein solches Event aus einer Komponente heraus zu erzeugen, wird der Dekorator *@Output()* verwendet. Die dazugehörige Property ist ein Event-Emitter, der Ereignisse auslösen kann (Listing 4).

Neben der Verwendung runder Klammern können Event-Bindings auch mit dem Ausdruck *on-{Event-Name}="{callback()}"* deklariert werden:

```
<car on-damaged="report(damage)"></car>
```

In der Dashboard-Komponente muss lediglich eine Methode ergänzt werden, die nach dem Auslesen des Events *(damaged)* ausgeführt wird (Listing 5). In diesem Fall wird im Dashboard die Anzahl der gemeldeten Schadensfälle zusammengezählt (Bild 4).

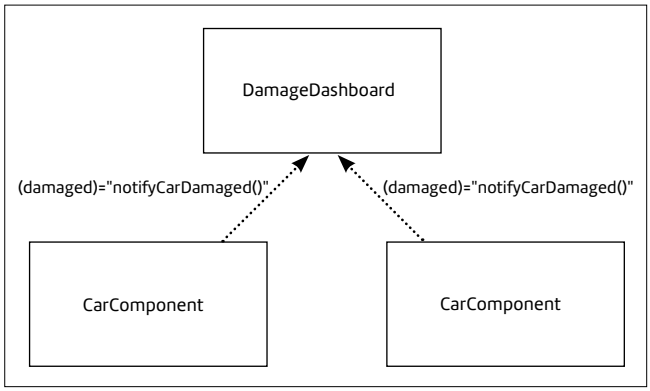
Two-way-Bindings

Um die Direktive *[(ng-model)]* zu verwenden, muss vorher das Modul *{ FORM_DIRECTIVES }* importiert werden.

Aus Sicht einer Komponente werden mit Property-Bindings schreibende und den Event-Bindings lesende Operationen spezifiziert. Wie in AngularJS 1.x ist es auch möglich, Zwei-Wege-Bindungen (Two-way-Bindings) zu realisieren. In der Template-Syntax von Angular 2.0 werden hierfür die Schreibweisen beider Binding-Arten kombiniert:

```
<input [(id)]="id">
```

Die eckigen Klammern legen fest, dass ein gegebener Wert an das *<input>*-Element gebunden wird. Die runden Klammern machen deutlich, dass Änderungen der Eigenschaft überwacht werden und diese mit Hilfe der Direktive *ng-model* in die Eigenschaft zurückschreiben werden.



Zustandsänderungen innerhalb einer Komponente publizieren (Bild 3)

Wie in den vorangehenden Beispielen gibt es auch hier eine alternative Schreibweise:

```
<input bindon-ng-model="id">
```

Die Zwei-Wege-Bindung lässt sich auch ohne *ng-model* realisieren. Das Markup wird so allerdings etwas komplexer:

```
<input
  [value]="id"
  (input)="id=$event.target.value">
```

Hierbei gibt *$event* Zugriff auf das auslösende Ereignis. Es ist ein natives JavaScript-Event. Daher kann dessen API verwendete werden, um auf das betroffene Element zuzugreifen und dessen Wert auszulesen *(id=$event.target.value)*.

Listing 4: car.component.ts

```
import { EventEmitter } from 'angular2/angular2';
@Component({ /* ... */ })
class CarComponent() {
  @Input() id:string;
  @Output() damaged:EventEmitter =
  new EventEmitter();

  reportDamage() {
    // Event auslösen
    this.damaged.next(this.id);
  }
}
```

Innerhalb eines Templates können Referenzen auf HTML-Elemente, Komponenten und Datenbindungen erzeugt werden, um mit ihnen zu arbeiten.

```
<input #id type="text"/>
{{ id.value }}
```

Das Binding *{{ id.value }}* macht deutlich, dass die lokale Referenz das HTML-Element referenziert und nicht nur dessen Wert. Anstatt der # können lokale Referenzen auch mit dem Prefix *var-* deklariert werden:

```
<input var-id type="text"/>
{{ id }}
```

Lokale Referenzen auf Komponenten unterscheiden sich in syntaktischer Hinsicht nicht von den HTML-Elementen. Zusätzlich können die Methoden der Komponente zur Interaktion genutzt werden:

```
<car #car></car>
<button (click)="car.getTankCapacity()">
Get tank capacity</button>
```

Lokale Referenzen können auch auf Objekte zeigen. Im folgenden Beispiel wird der Platzhalter *#c* genutzt, um für jedes Element der Liste *cars* die Komponente *Car* zu rendern:

```
<car *ng-for="#c in cars" [model]="c">
```

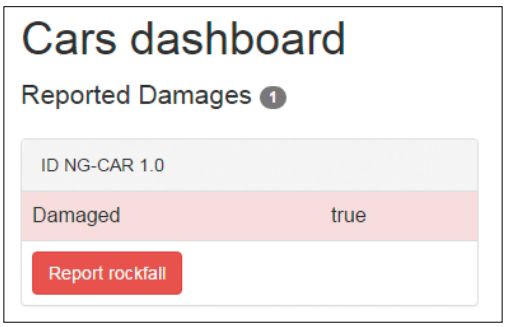
Listing 5: Auf Ereignisse einer Komponente reagieren

```
export default class DashboardComponent {
  /* ... */
  notifyCarDamaged() {
    this.totalDamages++;
  }
}
```

Bei dem Stern (*) vor der *ng-for*-Direktive handelt es sich übrigens um eine Kurzschreibweise.

Direktiven wie *ng-for*, *ng-if* und *ng-switch* werden zusammen mit einem Stern * verwendet. Diese Direktiven werden strukturelle Direktiven (Structural Directives) genannt, da sie DOM-Elemente hinzufügen oder entfernen:

```
<div *ng-if="totalDamages >
0">{{ totalDamages }}</div>
```



Anzeige der Anzahl aller erfassten Schäden (Bild 4)

In diesem Beispiel wird die Anzahl aller gemeldeten Schäden nur dann im Dashboard angezeigt, wenn deren Anzahl größer 0 ist.

Bei dem * handelt es sich um eine Kurzschreibweise, die das Schreiben des Templates vereinfachen soll. Sie wird als Micro-Syntax bezeichnet, da Angular 2.0 diesen Ausdruck interpretiert und wieder in die bekannten Bindings umsetzt. Beispielsweise ist auch folgende Verwendung der *ng-if*-Direktive zulässig:

```
<template [ng-if]="totalDamages > 0">
  <div>{{ totalDamages }}</div>
</template>
```

Angular übersetzt die Micro-Syntax in ein Property-Binding und umschließt das Template mit einem *<template>*-Tag. Dadurch entfällt der * vor dem *ng-if* (Tabelle 1).

Der Pipe-Operator

Pipes korrespondieren zu *filters* aus AngularJS 1.x und werden genutzt, um Daten zu für die Anzeige zu transformieren. Sie nehmen Eingabeargumente entgegen und liefern das transformierte Ergebnis zurück.

In einem Binding-Ausdruck werden sie durch das Symbol | (genannt Pipe) eingeleitet:

```
/* Der Wert von name wird in
Großbuchstaben ausgegeben */
<p>{{ id | uppercase}}</p>
```

Pipes können auch aneinandergehängt werden, um mehrere Transformationen durchzuführen:

```
<p>{{ id | uppercase |
lowercase}}</p>
```

Die Bezeichnung Elvis-Operator ist eine Ode an den Mythos, der sich damit befasst, ob Elvis tatsächlich tot ist oder nicht.

Der ? -Operator ist ein nützliches Instrument, um zu prüfen, ob ein Wert null oder undefined ist. Auf diese Weise können Fehlermeldungen bei der Template-Erzeugung vermieden werden:

```
<p>{{ car?.driver }}</p>
```

Hier wird geprüft, ob das Objekt *car* existiert. Wenn ja, wird der Namen des Fahrers ausgegeben. Der ?-Operator funktioniert ebenfalls in komplexeren Objektbäumen:

```
<p>{{ car?.driver?.licences?.B1 }}</p>
```

Auch wenn sich die Syntax zu Beginn ungewohnt ist, handelt es sich hierbei um valides HTML. In der HTML-Spezifikation des W3C heißt es: »Attribute names must consist of one or more characters other than the space characters, U+0000 NULL, """, "'", ">", "/", "=", the control characters, and any characters that are not defined by Unicode.«

Polymer-Webkomponenten nutzen

In AngularJS 1.x ist Entwicklungsaufwand nötig, um Webkomponenten anderen Bibliotheken integrieren zu können. Es müssen Direktiven geschrieben werden, um Angular die Statusänderungen der Fremdkomponenten mitzuteilen.

Mit Angular 2.0 ist diese Arbeit nicht mehr nötig. Es wird nicht mehr unterschieden, ob es sich um ein natives Browserelement oder eine Web Component handelt. Angular hat nur Kenntnis davon, dass es an bestimmten Stellen im DOM Elemente instanzieren muss und es Eigenschaften schreiben sowie Event Listener erzeigen soll. Das ermöglicht beispielsweise die direkte Verwendung der Komponente *google-youtube* aus dem Polymer-Projekt:

```
<google-youtube
  #player
```

Tabelle 1: Übersicht der Binding-Typen in Angular 2.0

Binding-Typ	Anwendung	Beispiel
Property	Element Event Component Event Directive Event	`<car [id]="model.id"></car>` `<img [src]="model.imgUrl"></img>`
Event	Element Event Component Event Directive Event	`<car (damaged)="notifyCarDamaged()"></car>` `<button (click)="save()"></button>`
Two-way	Directive-Event-Property	`<input [(ng-model)]="car.driver">`
Attribute	Attribut	`<input [disabled]="model == null">` `<button [attr.aria-label] = "actionName">`
Class	Class-Property	`<tr [class.success]="model?.hasDamage == false">`
Sytle	Style-Property	`<span [style.color]="car.HasDamage : 'red' ? 'black'"></span>`

Zusammenfassung

Die wichtigsten Fakten zur Template-Syntax auf einem Blick:

- Input- und Output-Properties beschreiben das API einer Komponente.
- Über Inputs fließen Daten in die Komponente hinein.
- Inputs werden über Property-Bindings aktualisiert *([property])*.
- Über Outputs fließen Daten aus der Komponente heraus.
- Outputs werden mit Hilfe von Event-Bindings abonniert *(event)*.
- Ein Property-Binding und Event-Binding können kombiniert werden, um ein Two-way-Binding zu beschreiben *(twoWay)*.

```
  [video-id]="videoId">
</google-youtube>

<button (click)="player.play()"></button>
<button (click)="player.pause()"></button>

@View({ /* … */})
export default class DashboardComponent {
  /* ... */
  videoId: string;
  constructor() {
    /* ... */
    this.videoId = "ewxEFdMPMFO";
  }
```

Alle im Artikel beschriebenen Konzepte können hier nahtlos verwendet werden. Anhand der Online-Dokumentation von *google-youtube* ist bekannt, welche Eigenschaften und Aktionen zur Verfügung stehen. Das Element-Attribut *video-id* kann über ein Property-Binding gesetzt werden (*[video-id]*). Wird der Komponente eine gültige ID eines Videos von YouTube übergeben, initialisiert sich der Video-Player selbstständig und kann verwendet werden.

Unter Verwendung der Referenz *#player* können die Aktionen der Webkomponente von anderen Webelementen gesteuert werden. Angular stellt über die Template-Syntax ein einheitliches API zur Verfügung, das auf jeder Web Component angewendet werden kann.

Fazit

Angular 2.0 bricht die Template-Syntax in mehrere Konzepte auf. Der Datenfluss zwischen Komponenten wird dadurch konkret definiert. Daher ist es mit einem Blick auf ein Template möglich, zu erkennen, wie sich eine Komponente verhält. Somit können im Unterschied zur Vorgängerversion Templates in Angular 2.0 diffiziler und genauer beschrieben werden.

Allerdings sind auch mehrere Möglichkeiten vorhanden, Templates und Bindings zu definieren. Daher ist es ratsam, sich im Team auf jeweils eine der angebotenen Schreibweisen zu einigen, um ein vertrautes und homogenes Bild im Markup zu schaffen.

Im nächsten Artikel in der Ausgabe 2/2016 der **web & mobile developer** werden die Themen Dependency Injection und Unit-Testing mit Angular 2.0 behandelt. Denn wie in AngularJS 1.x können bei dessen ambitioniertem Nachfolger Komponenten und Dienste über Angulars integrierten IoC-Container miteinander kombiniert werden und dennoch für sich isoliert getestet werden. ■

Links zum Thema

- Angular 2.0 – 5 Min Quickstart
  https://angular.io/docs/ts/latest/quickstart.html
- Angular 2.0 Template Syntax
  http://victorsavkin.com/post/119943127151/angular-2-template-syntax
- ng-conf 2015 Keynote 2
  https://www.youtube.com/watch?v=-dMBcqwvYA0
- Felipe Coury, Nate Murray, Carlos Taborda
  ng-book 2 – The Complete Book on AngularJS 2
  https://www.ng-book.com/2/
- angular.io – Template-Syntax
  https://angular.io/docs/ts/latest/guide/template-syntax.html
- Template-Syntax demystified
  http://blog.thoughtram.io/angular/2015/08/11/angular-2-template-syntax-demystified-part-1.html
- Angular 2 Data Flow – Jeff Cross, Rob Wormald and Alex Rickabaugh
  https://youtu.be/bVI5gGTEQ_U
- W3C – HTML: The Markup Language (an HTML language reference)
  http://www.w3.org/TR/html-markup/syntax.html

Johannes Hoppe
ist selbstständiger IT-Berater, Softwareentwickler und Trainer. Er arbeitet derzeit als Architekt für ein Portal auf Basis von .NET und AngularJS.
http://blog.johanneshoppe.de

Gregor Woiwode
ist als Software-Entwickler im Bereich des Competitive Intelligence bzw. Enterprise Knowledge Managements für ein Softwareunternehmen in Leipzig tätig. Er veranstaltet Trainings AngularJS.
www.woiwode.info/blog

DOMAINS | E-MAIL | HOSTING | SHOPS | **SERVER**

1&1 CLOUD SERVER

TEST THE BEST!

TOP-PERFORMER CLOUD SPECTATOR

Powered by intel Cloud Technology

Easy to use – ready to cloud.

Die 1&1 Cloud Server sind unschlagbar in ihrer Performance bei CPU, RAM und SSD! Realisieren Sie Ihre Cloud-Projekte mit der perfekten Kombination aus Flexibilität und leistungsstarken Features.

- ✓ **Load Balancing**
- ✓ **SSD Storage**
- ✓ **Minutengenaue Abrechnung**
- ✓ **Intel® Xeon® Prozessor E5-2660 v2 und E5-2683 v3**

1 Monat kostenlos!

Danach ab 4,99 €/Monat.*

Internet made in Germany

1 TEST 30 TAGE AUSPROBIEREN

1 KLICK UP- ODER DOWN-GRADEN

1 ANRUF EXPERTEN-RAT

DE: 02602 / 96 91
AT: 0800 / 100 668

1&1

1und1.info

*1&1 Cloud Server 1 Monat kostenlos. Danach ab 4,99 €/Monat. Keine Einrichtungsgebühr. Keine Mindestvertragslaufzeit. Preise inkl. MwSt. Intel®und das Intel® Logo sind eingetragene Marken der Intel Corporation in den USA und anderen Ländern. 1&1 Internet SE, Elgendorfer Straße 57, 56410 Montabaur.

AUTOMATISIERTES TESTEN VON WEBSEITEN

Komplexe Applikationen

Komplexe Webseiten und Single Page Applications benötigen einen hohen Testaufwand.

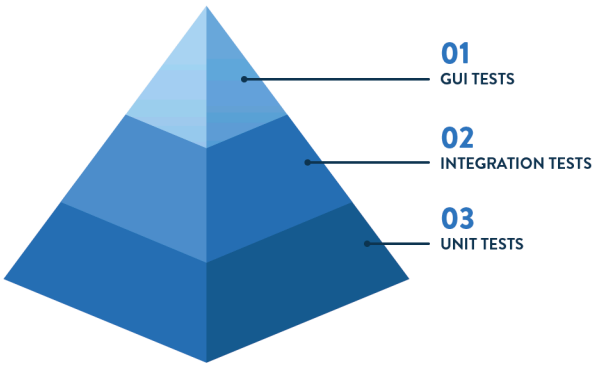
Webseiten haben sich im Lauf der Zeit von einfachen Textdokumenten mit Bildern und einem simplen Layout zu komplexen Webapplikationen entwickelt, die individuell auf die Interaktion des Benutzers reagieren. Heutzutage sind viele Webseiten sogenannte Single Page Applications wie zum Beispiel Google Documents oder der Spotify Web Player.

Solche komplexen Applikationen sind entsprechend fragil und benötigen einen hohen Testaufwand, um sicherzustellen, dass die Applikation in allen unterstützten Browsern und auf allen Endgeräten auch so funktioniert, wie es die Entwickler beabsichtigt haben.

In der klassischen Programmierung sind Tests gang und gäbe. Durch die zunehmende Professionalisierung im Bereich Frontend- und User-Interface Development und die immer komplexeren Abläufe und Features innerhalb moderner Webapplikationen steigt dementsprechend auch das Interesse und Bedürfnis, Tests durchzuführen.

Der Browser als solches ist keine Plattform, die fest definiert ist. Es handelt sich vielmehr um eine Plattform, die viele Unbekannte aufweist. Einerseits haben wir hochmoderne Browser wie Google Chrome oder Firefox, die Rapid-Release-Zyklen bieten, um stets auf dem neuesten Stand zu sein, und entsprechend neue Features in monatlicher Taktung zur Verfügung stellen.

Auf der anderen Seite werden auf PCs oftmals aber auch noch veraltete Browser verwendet, denen existenzielle Funktionen fehlen oder, die ein ganz anderes Verhalten an den Tag legen wie moderne Browser. Um diesem Problem Herr zu werden, gibt es zwei Wege.



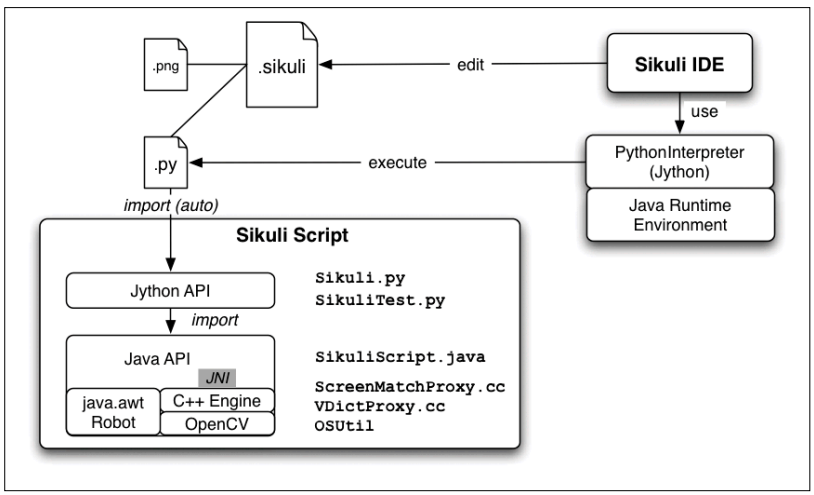
Varianten: Die verschiedenen Arten von Tests für Webapplikationen (Bild 1)

Progressive Enhancement

Einerseits kann man fehlende Funktionen über sogenannte Polyfills nachrüsten. Dies geschieht üblicherweise über JavaScript-Bibliotheken, die die Funktionen nachbilden.

Eine wesentlich elegantere Herangehensweise, die zum Beispiel von Jeremy Keith vertreten wird, ist das sogenannte Progressive Enhancement.

Man identifiziert dabei die Kernfunktionalität der Webapplikation beziehungsweise der Komponente und setzt dann die simpelste Technologie ein, um das gewünschte Feature umzusetzen. Später baut man dieses Feature mit modernen Technologien entsprechend aus. So wird sichergestellt, dass einerseits veraltete Browser weiterhin die Webapplikation nutzen können, Benutzern mit modernen Browsern aber ein Mehrwert geboten wird. Leider machen diese beiden Ansätze die Webapplikation nicht stabiler, sondern im Gegenteil wird so in den verschiedenen Browsern häufig ein abweichendes Verhalten verursacht (Bild 1).



Ablaufdiagramm von Sikuli Script (Bild 2)

Manuellen Testaufwand verringern

Durch diese Vielzahl an Kombinationen und unterschiedlichen Feature-Sets pro Browser wird der manuelle Testaufwand so groß, dass es kaum noch möglich ist, bei tiefgreifenden Änderungen einen kompletten Test der gesamten Webapplikation durchzuführen. An dieser Stelle kommen automatisierte Tests zum Tragen. Doch welche Art von Tests eignet sich für den Anwendungsfall eine Webseite oder, genauer gesagt, um das User Interface zu testen?

In der klassischen Programmierung setzt man oft auf Unit-Tests. Unit-Tests sind bestens geeignet, um Teilbereiche einer Webapplikation zu testen. So ist es möglich, sehr granulare Tests zu erstellen, die sicherstellen, dass die existenzielle Business-Logik und die entsprechend definierten Workflows funktionieren.

Ein weiterer Pluspunkt von Unit-Tests ist, dass diese eine gewisse Sicherheit bei Änderungen geben. So kann bei tiefgreifenden Änderungen oder größeren Umbauarbeiten sichergestellt werden, dass alles weiterhin so funktioniert wie erwartet. Es besteht kein Zweifel darin, dass Unit-Tests die Qualität von unzähligen Software-Projekten verbessern. Aber machen Unit-Tests andere Tests überflüssig? Nein, keinesfalls. Man braucht mehr als nur Unit-Tests, um eine professionelle Softwarequalität zu erreichen.

Üblicherweise schreiben Entwickler ihre eigenen Unit-Tests selbst, was aber zu Problemen führen kann. Resultierend daraus, dass Entwickler die Tests mit der gleichen Denkweise schreiben wie den Code, werden Probleme, die dem Entwickler selbst nicht auffallen, auch nicht vom Test abgedeckt. Um einen Unit-Test zu schreiben, empfiehlt es sich daher, die Tests ebenfalls einem Code-Review zu unterziehen, sodass ein unvoreingenommener Entwickler den Unit-Test aus einer anderen Perspektive begutachtet und entsprechend andere Probleme sehen kann.

Benutzer verbringen ihre gesamte Zeit ausschließlich mit der Interaktion mit dem User Interface der Webapplikation. Unit-Tests sind aber Backend-Tests, die zum Beispiel prüfen, ob eine Kalkulation richtig ist. Sie können nicht sicherstellen, ob das Resultat dieser Kalkulation auch korrekt dargestellt wird. Des Weiteren können zum Beispiel Probleme wie fehlerhafte Icons, falsch platzierte Call-to-Action-Buttons, verschobene Feld-Labels oder abgeschnittene Texte nicht erkannt werden – hierfür benötigt man User-Interface-Tests.

User-Interface-Tests – welche Möglichkeiten gibt es?

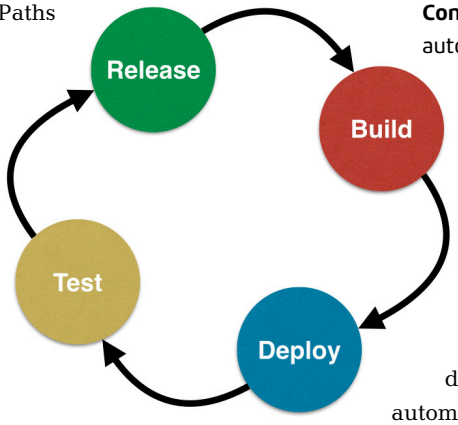
Es gibt verschiedenste Ansätze, wie man UI-Tests schreiben kann, und entsprechend viele Tools. Eines der wohl bekanntesten ist Selenium. Selenium ermöglicht es, Browser zu automatisieren. So kann man Workflows abbilden und prüfen, ob diese nach einer Refaktorierung weiterhin funktionieren. Selenium arbeitet auf Basis sogenannter XPaths.

XPaths repräsentieren einen eindeutigen Pfad innerhalb einer DOM-Struktur, um Elemente zu selektieren. Auf Basis der selektierten Elemente können dann Assoziationen durchgeführt werden. Zum Beispiel um zu prüfen, ob das Label des Feldes korrekt ist. Selenium benötigt zum Ausführen der Tests jeweils einen sogenannten Testrunner, der den Browser öffnet und die Steuerbefehle durchführt.

Alternativ ist es möglich, die Tests in einem Headless-Browser wie zum Beispiel PhantomJS durchzuführen. So ist es möglich, die Tests nicht nur auf seinen eigenen Rechnern, sondern auch auf einem Continous-Integration-Server durchzuführen. So kann man beispielsweise bei jedem Commit in der Versionskontrolle einen Test-Run auslösen und somit sicherstellen, dass der Commit keine Probleme verursacht. ▶

Leider hat Selenium aufgrund der XPaths aber auch eine Kehrseite. Falls sich ein Selektor ändert, eine andere Klasse auf ein Element gelegt wird oder ein Element verschoben wird, dann muss zudem der Test manuell angepasst werden.

Wir hatten vor allem bei unserer Administrationsoberfläche mit XPaths zu arbeiten, da sich die DOM-Struktur aufgrund des verwendeten JavaScript-Frameworks Ext.JS stetig geändert hat.



Continous Deployment für Webseiten mit automatisierter Testdurchführung (Bild 3)

Visuelles Testen von Webapplikationen

Wir haben uns angesichts dieses Problems auf die Suche nach einem anderen Tool gemacht, das unabhängig von der Struktur ist, und sind dabei auf SikuliX gestoßen. SikuliX verfolgt den gleichen Ansatzpunkt wie Selenium und automatisiert Browser.

Doch SikuliX geht noch einen Schritt weiter und kann alles automatisieren, was auf dem Monitor zu sehen ist. Zur Identifizierung von Elementen wird hier nicht auf XPaths oder CSS-Selektoren, sondern auf Screenshots gesetzt. Ja, Sie haben richtig gelesen. SikuliX verwendet intern OpenCV (Open Computer Vision), um eine Bild-Erkennung durchzuführen (Bild 2).

SikuliX selbst stellt eine Python Library zur Verfügung, um Workflows und zum Beispiel Klickwege zu skripten. Um aber mit SikuliX wirkliche Tests zu schreiben, benötigen wir eine weitere Komponente, das sogenannte Robot Framework. Hierbei handelt es sich um ein Testautomatisierungs-Framework, um Akzeptanz-Tests zu schreiben. Die Tests selbst werden in einer tabellarischen Daten-Syntax und mit Keyword-basiertem Ansatz erweitert. So kann die Basis-Funktionalität des Frameworks erweitert werden. Nützlich ist dies, wenn man gewisse Workflows, wie zum Beispiel einen Benutzer-Login, der in mehreren Tests zum Einsatz kommt, automatisieren möchte. Einen typischen Test zeigt Listing 1.

Das Robot Framework sorgt auch dafür, dass wir unsere Tests in die beiden Testsuits Storefront und Administration aufteilen können (Bild 3).

Listing 1: Test

```
*** Settings ***
Library CalculatorLibrary

*** Test Cases ***
Addition
  Given calculator has been cleared
  When user types "1 + 1"
  and user pushes equals
  Then result is "2"

*** Keywords ***
Calculator has been cleared
  Push button    C

User types "${expression}"
  Push buttons    ${expression}

User pushes equals
  Push button    =
```

Tests automatisiert ausführen lassen

Nachdem wir uns für SikuliX in Kombination mit dem Robot Framework entschieden haben und eine Test-Suite mit knapp 300 Tests aufgebaut hatten, wollten wir natürlich auch sicherstellen, dass wir jeweils täglich feststellen können, ob alle Refrakturierungen und Änderungen am Frontend weiterhin funktionieren und kein Bruch entstanden ist.

Da die Ausführung der Tests circa 60 Minuten pro Browser dauern, haben wir uns für einen Nightly Run entschieden, der jeweils um drei Uhr nachts ausgeführt wird. Die jeweiligen Browser haben wir in VMs zur Verfügung gestellt. Das Robot Framework stellt Reporter in verschiedensten Formaten bereit, sodass einer Integration in Jenkins oder Bamboo nichts mehr im Wege steht.

Fazit

Tests sind wichtig, um eine professionelle Codequalität zu liefern und sicherzustellen, dass die Software jederzeit veröffentlich werden kann. Außerdem geben Tests Sicherheit bei der Entwicklung an bestehendem Code. Doch es reicht nicht, nur Unit-Tests durchzuführen. Integrationstests wie UI-Tests sind ebenso Bestandteil wie Regressions-Tests. Im Bereich Unit-Tests gibt es unzählige Ansätze, Frameworks und Tools – doch UI-Tests sind aktuell eher noch eine Individual-Lösung. Durch die Verwendung des Robot Frameworks in Kombination mit SikuliX ist es aber möglich, eine universelle Lösung zu finden, die auch dann funktioniert, wenn die DOM-Struktur dynamisch generiert wird. ■

Stephan Pohl
ist Teamleiter Web-Development bei der Shopware AG und Spezialist auf den Gebieten JavaScript, CSS und HTML. Er fungiert als Schnittstelle zwischen den Abteilungen Design und Entwicklung.
https://de.shopware.com

Developer Newsletter

Top-Informationen für Web- und Mobile-Entwickler.
Klicken. Lesen. Mitreden.

Jetzt kostenlos anmelden:

webundmobile.de

twitter.com/webundmobile

facebook.de/webundmobile

gplus.to/webundmobile

BIBLIOTHEK REDUX

Unidirektional

Redux ist eine Bibliothek zur Zustandsverwaltung in JavaScript-Anwendungen.

Funktionale Programmierkonzepte halten mehr und mehr Einzug im Mainstream. Facebooks React übt auch im Umfeld der Frontend-Entwicklung mit JavaScript einen wachsenden Einfluss darauf aus. Doch React ist in erster Linie eine View-Library – es bietet für sich alleine betrachtet kaum Lösungen zur Verwaltung des Anwendungszustands. Mit Flux hat Facebook seine Idee dazu publik gemacht. Dan Abramovs Redux ist eine Variante davon, die sich aus der Masse an Flux-Implementierungen sehr positiv abhebt.

Die Verwaltung des Anwendungszustand wird bei zunehmend komplexen Anwendungen immer schwieriger. Unter Anwendungszustand versteht man die Gesamtmenge aller Daten, die den Zustand der Anwendung zu einem bestimmten Zeitpunkt ausmachen. Dazu gehören die momentan geladenen oder angezeigten Inhalte, die Auswahl-, Anzeige-, Ausklappzustände der verschiedensten UI-Elemente und vieles mehr. Auch der Zustand von Kommunikationsvorgängen und -verbindungen (verbunden oder nicht verbunden, Anfrage gesendet, Anfrage empfangen) wird häufig benötigt.

Die Verwaltung dieses Anwendungszustands über die Laufzeit des Programms hinweg kann eine große Herausforderung sein. Oft wird er ohne ein einheitliches Konzept gepflegt und dem Entwickler ist teilweise gar nicht bewusst, welchen Anwendungszustand sein Programm erzeugt. Seit es Computerprogramme gibt, hat es unzählige Versuche gegeben, um die Verwaltung des Anwendungszustands zu vereinfachen.

Probleme mit MVC

Bis heute ist das Model-View-Controller-Konzept eine der am meisten verbreiteten Ideen, um Separation of Concerns in einer Anwendungsarchitektur umzusetzen. Man kann es deshalb auch als einen Versuch ansehen, die Verwaltung des Anwendungszustands zu vereinfachen, indem klare Zuständigkeiten ermittelt werden.

Von MVC-Frameworks wie Backbone kennt man bereits eine eigene Abstraktionsschicht mit sogenannten Model-Objekten. Der Sinn besteht darin, dass üblicherweise das Observer-Designmuster angewendet wird: Die View beobachtet die Model-Objekte und bekommt auf diese Weise Änderungen am Anwendungszustand mit.

Bei komplexen MVC-Anwendungen mit verschachtelten und hierarchischen Model-Objekt-Bäumen kommt es auch vor, dass Model-Objekte andere Model-Objekte beobachten und auf diese Weise mehrstufige Benachrichtigungskaskaden entstehen können.

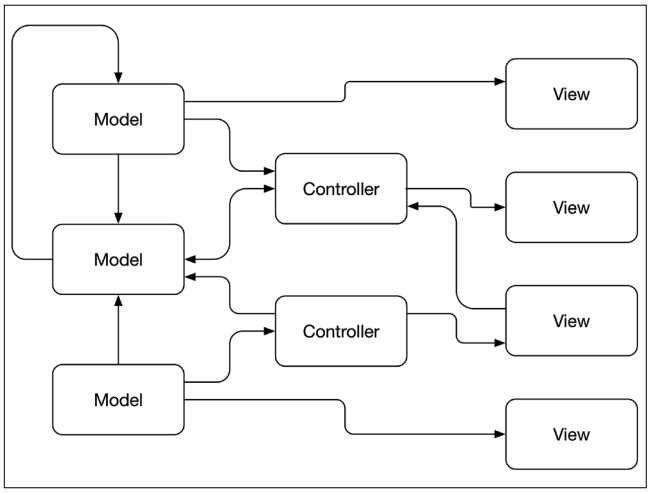
Dies kann jedoch zu Problemen führen: Bild 1 zeigt wie mit zunehmender Komplexität einer MVC-Anwendung schnell der Überblick über die Beziehungen der einzelnen Komponenten verloren gehen kann. Besonders kritisch sind dabei auch immer wieder mehrfache Observer-Beziehungen innerhalb der Model-Schicht. Oft beginnen Programmierer, den Anwendungszustand noch um zusätzliche Flags anzureichern, die verhindern sollen, dass abhängig vom Anwendungszustand bestimmte Teilnehmer durch signalisierte Änderungen beeinflusst werden. Updates werden mitunter doppelt und dreifach durchgeführt.

Ein weiteres Problem ist, dass die Model-Schicht im Normalfall veränderlichen Zustand enthält, der noch dazu asynchron modifiziert wird. Race Conditions und Fehler aufgrund von nichtdeterministischem Timing-Verhalten sind damit schwer nachvollziehbar.

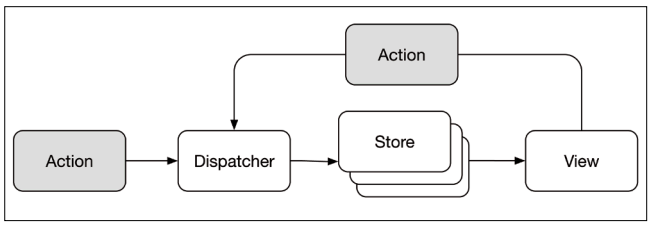
Unidirektionaler Datenfluss

Als Lösung für die mit dem MVC-Modell einhergehenden Probleme hat Facebooks Entwicklungsabteilung die Anwendungsarchitektur Flux entwickelt (Bild 2). Zuerst nur als allgemeine Beschreibung; später folgte eine Referenzimplementierung des Konzepts. Spätestens seitdem gibt es eine Vielzahl an Implementierungen. Allen gemein ist die Grundidee eines unidirektionalen Datenflusses.

Der Zustand der Anwendung wird in sogenannten Stores gespeichert. Aus diesen wird (oder werden) die View(s) erzeugt. Hier zeigt natürlich eine View-Library wie Facebooks React ihre Stärken, denn React zeichnet sich insbesondere dadurch aus, dass es die View bei einem geänderten Zustand sehr effizient neu rendern kann, indem es sein virtuelles



Model-View-Controller-Problematik: Die Komplexität einer MVC-Anwendung (Bild 1)



Facebooks Flux: Lösung für die mit dem MVC-Modell einhergehenden Probleme (Bild 2)

DOM mit dem realen DOM vergleicht und die notwendigen Änderungsoperationen berechnet.

Jede Änderung des Zustands in den Flux-Stores muss über sogenannte Actions durchgeführt werden, welche die Art der Änderung beschreiben. Der View selbst kann Actions auslösen, die wiederum über den Dispatcher an die Stores verteilt werden. Die Daten fließen in dieser Architektur also stets in eine Richtung.

Redux ist eine Bibliothek zur Verwaltung von Zustand in JavaScript-Anwendungen. Der Entwickler Dan Abramov hat sie im Rahmen der Vorbereitung zu seinem Vortrag »Hot Reloading with Time Travel« auf der React Europe Konferenz geschrieben. Seitdem hat er für diese kleine Bibliothek von vielen Seiten großes Lob eingeheimst.

Redux gilt im allgemeinen als eine Flux-Variante, obwohl es auch einzelne Stimmen gibt, die Redux nicht als eine Implementierung von Flux ansehen, weil das Konzept des Dispatchers fehlt und es auch lediglich einen einzigen Store gibt. Der Hauptgrund, warum diese Konzepte fehlen, ist jedoch die radikale Reduktion auf das absolut Wesentliche.

Redux ist Reduced Flux

Denn Redux reduziert das Flux-Konzept auf drei Bedingungen:

1. **Single Source of Truth:** Der vollständige Anwendungszustand wird in einem einzigen Store als ein einziger Objektbaum gehalten. Trotz dieser Vereinfachung wird auch bei Redux nicht alles an einem Ort durcheinandergeworfen. Unterschiedliche Aspekte können schließlich auch einfach in unterschiedlichen Teilbäumen gegliedert sein. Trotzdem vereinfacht der einfache Objektbaum Operationen, die auf dem gesamten Anwendungszustand zugreifen wollen.
2. **Zustand ist read-only:** Der im Store gehaltene Zustand wird – wie bei Flux üblich – ausschließlich über Actions modifiziert. Auf diese Weise gehören Probleme wie Race Conditions der Vergangenheit an, weil diese Actions stets nacheinander angewandt werden. Der aktuelle Zustand im Redux-Store wird niemals direkt modifiziert – stattdessen wird stets ein neuer Zustand erzeugt, der den aktuellen dann ersetzt.
3. **Veränderung des States heißt Transformation durch Reducer:** Die Berechnungsvorschrift eines neuen Zustands wird mittels der sogenannten Reducer realisiert. Eine Redux-Anwendung hat einen oder mehrere Reducer. Diese berechnen anhand des aktuellen Zustands und einer Action den neuen Zustand. Da der State stets neu berechnet wird, kann es nicht passieren, dass die Anwendung ungültige Zwischenzustände einer direkten Manipulation sieht.

Dan Abramov veröffentlichte auf Twitter als Spielerei eine vollständige Redux-Implementierung in etwa 100 Zeilen JavaScript – was deutlich macht, wie stark das Konzept reduziert werden konnte.

Funktionale Konzepte

Ein Grund für die Kompaktheit der Bibliothek ist die konsequente Anwendung funktionaler Konzepte: Wenn eine Funktion zur Abbildung eines Konzepts ausreicht, dann muss man dafür kein Objekt mit einer Methodenschnittstelle schaffen. Für Neulinge funktionaler Programmierung erscheint das oft zuerst ungewohnt. Tatsächlich ist in der JavaScript-Community jedoch spätestens seit dem Erfolg von React ein starker Trend funktionaler Programmierkonzepte sichtbar. Deshalb hier die grundlegenden Konzepte:

Die Funktion ist sicherlich das am weitesten bekannte Konstrukt. Man definiert sie mit dem *function*-Schlüsselwort, und sie kann Parameter empfangen und liefert einen Rückgabewert. Bei funktionaler Programmierung gilt üblicherweise, dass eine Funktion nur von ihren Parametern abhängt und immer ein Resultat zurückliefert. Für die gleichen Parameter sollten stets das gleiche Resultat zurückgegeben werden. Wenn das nicht so ist, dann würde die Funktion von Seiteneffekten abhängen und wäre nicht mehr pur. Eine der wesentlichen Merkmale funktionaler Programmierung ist die konsequente Vermeidung von Seiteneffekten. Hier eine einfache Funktion:

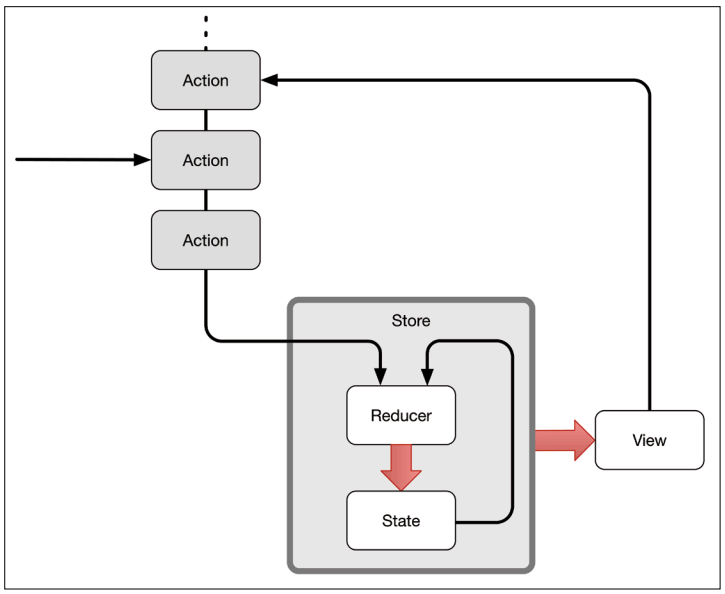
```
function plus (a,b) {
  return a+b;
}
```

Nahezu trivial ist hier zu erkennen, dass die Funktion lediglich von ihren beiden Parametern *a* und *b* abhängt und dass bei gleichen Werten auch stets dasselbe Ergebnis berechnet wird. Ebenfalls eine Javascript-Funktion, aber mit Seiteneffekt und deshalb nicht funktionaler Programmierung folgend:

```
let counter = 0;
function count () {
  return counter++;
}
```

Die Funktion hängt von einer Zustandsvariable in ihrem lexikalischen Gültigkeitsbereich ab. Innerhalb der Funktion wird diese Zustandsvariable bei jedem Aufruf modifiziert (Seiteneffekt). Die Funktion liefert mit jedem Aufruf ein anderes Ergebnis und das, obwohl sie nicht einmal Parameter besitzt. Streng funktional würde eine Funktion ohne Parameter immer den gleichen Wert liefern.

Man beachte: Damit sind die allermeisten Methoden von Objekten nicht streng funktional – denn entweder sie hängen vom internen Zustand ihres Objekts ab, oder sie beeinflus- ▶



Redux: Bei Flux werden Actions durch sogenannte Action Creators erzeugt (Bild 3)

sen selbst Objektzustände durch Seiteneffekte. Objektorientierte Programmierung ist inhärent inkompatibel mit den Konzepten streng funktionaler Programmierung.

Funktionales Programmierkonzept

Ein sehr gängiges funktionales Programmierkonzept sind Funktionen höherer Ordnung. Das klingt zwar etwas geschwollen, heißt aber einfach nur, dass eine solche Funktion eine andere Funktion als Parameter erhält. Die bekanntesten Vertreter in JavaScript sind *Array.prototype.map*, *Array.prototype.filter* oder *Array.prototype.reduce*.

Mit *map()* kann man eine übergebene Funktion auf jedes Element eines Arrays anwenden. Das Ergebnis des *map()*-Aufrufs ist ein neues Array mit dem jeweiligen Resultat der übergebenen Funktion:

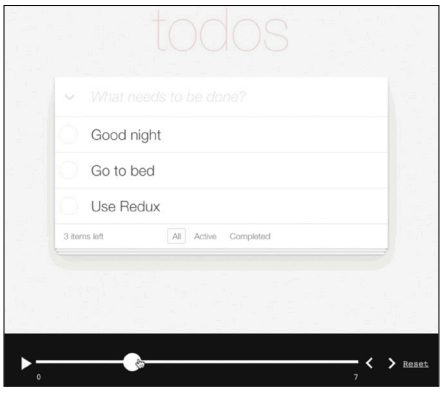
```
[1,2,3,4,5].map(x=>x+1)
=> [2,3,4,5,6]
```

Mit *filter()* kann man aus den Elementen eines Arrays jene herausfiltern, für welche die übergebene (Prädikat-) Funktion *true* zurückliefert:

```
[1,2,3,4,5].filter(x=>(x%2)===0)
=> [2,4]
```

Reduce ist neben *map()* und *filter()* das wohl bekannteste Konzept funktionaler Programmiersprachen. Typische Anwendungsfälle sind die Summe oder Produkte der Elemente eines Arrays:

```
[1,2,3,4,5].reduce((a,b)=>a+b, 0);
=> 15
```



Time Travel Debugger mit einer Redux-Implementierung des bekannten Todo-MVC-Projekts (Bild 4)

```
[1,2,3,4,5].reduce((a,b)=>a*b, 1);
=> 120
```

Aber auch String-Konkatenation kann man damit erreichen:

```
["1", "2", "3", "4", "5"].reduce((a,b)=>a+","+b,
"")
=> "1,2,3,4,5"
```

Das letzte hier vorgestellte funktionale Konzept ist die Kombination mehrerer Funktionen in eine. Eines der gängigsten Beispiele ist *compose*:

```
function compose2 (f,g) {
  return (...args) => f(g.apply(args))
}
function compose (...fns) {
  return fns.reduce(compose2);
}
let add1 = x=>x+1;
let double = x=>x*2;
// f(x) = (x*2)+1
f = compose(add1, double)
```

Das Beispiel zeigt, wie man *compose* auf einfache Weise implementieren könnte. Die Hilfsfunktion *compose2* erhält zwei Funktionen als Parameter. Die erste Funktion *(f)* wird mit dem Ergebnis des Aufrufs der zweiten Funktion aufgerufen. Man könnte auch sagen, dass *f* zu einer Wrapper-Funktion wird.

Die Implementierung der Funktion *compose* wendet nun diesen kleinen Wrapper-Generator mit *reduce()* auf eine Liste von Funktionen an. Jede einzelne Funktion wird um ihren Nachfolger gewickelt, und ganz innen befindet sich der Aufruf der letzten Funktion der Liste.

Funktionsweise von Redux

Bild 3 zeigt, wie Redux funktioniert. Wie bei Flux werden Actions durch sogenannte Action Creators erzeugt, doch es gibt keinen eigenen Dispatcher. Stattdessen werden sie im Store zusammen mit dem aktuellen Zustand an die Reducer-Funktion des Stores übergeben. (Die Methode des Stores, mit dem man eine Action übergibt, heißt jedoch auch *dispatch()*). Das Resultat der Reducer-Funktion ist der neue aktuelle Zustand des Stores. Statt die Reducer-Funktion also auf ein Array von Elementen anzuwenden, wird bei Redux die Reducer-Funktion auf die zeitlich nacheinander ankommenden Actions angewandt, um kontinuierlich den jeweils aktuellen Zustand auf den nächsten zu reduzieren.

Da der Zustand niemals direkt manipuliert wird, ist es ein Leichtes, vorherige Zustände wiederherzustellen – es

gehört nicht mehr dazu, als sich eine Referenz auf den vorherigen Zustand zu merken. Bild 4 zeigt einen sogenannten Time Travel Debugger mit einer Redux-Implementierung des bekannten TodoMVC-Projekts. Durch Ziehen an dem Slider auf der Unterseite des UI kann man verschiedene Zustände des UI wiederherstellen.

Installation von Redux

Wie viele Projekte im React-Umfeld ist der gängige Installationpfad von Redux auch per Node Package Manager (npm). Alle anderen Möglichkeiten sind dagegen umständlich oder zumindest nicht empfehlenswert. Man beginnt sein eigenes Projekt im einfachsten Fall, indem man ein Verzeichnis anlegt und dieses mit *npm* initialisiert:

```
$ mkdir redux-demo

$ cd redux-demo/
$ npm init
...
```

Für React benötigt man üblicherweise einen Compiler, der JSX-Code übersetzen kann. Seit einiger Zeit setzt Facebook dabei auf Babel, was den besonderen Vorteil hat, dass es gleich noch eine ganze Reihe weiterer Features von ECMAScript 2015 und teilweise sogar einige der Vorschläge für ES2016 mitbringt. Die *package.json* enthält alle für das Beispielprojekt notwendigen Abhängigkeiten (Listing 1).

Listing 1: package.json

```
{
  "name": "redux-artikel",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\"
    && exit 1"
  },
  "author": "",
  "license": "ISC",
  "dependencies": {
    "react": "^0.14.2",
    "react-dom": "^0.14.2",
    "react-redux": "^4.0.0",
    "redux": "^3.0.4",
    "redux-thunk": "^1.0.0"
  },
  "devDependencies": {
    "babel-core": "^5.8.22",
    "babel-loader": "^5.3.3",
    "webpack": "^1.12.3"
  }
}
```

Nachdem die *package.json* ins Projektverzeichnis kopiert wurde, installiert man die Abhängigkeiten durch folgenden Aufruf:

```
npm install
```

Unter den Abhängigkeiten befinden sich die Module für React und ReactDOM. Weiterhin auch Redux in Form des npm-Pakets redux. Das Paket react-redux ergänzt Redux um Werkzeuge zur Anbindung an React. Eine Redux-Erweiterung mit dem Namen redux-thunk wird später noch interessant, wenn es um asynchrone Action Creators geht.

Die Entwickler-Abhängigkeiten enthalten den Babel-Compiler, webpack und den babel-loader, den webpack benutzt, um JavaScript-Dateien mit Babel zu übersetzen. Die Konfiguration für webpack zeigt Listing 2.

Die Quellen der Anwendung liegen in einem Unterverzeichnis *app/*. Dort ist die Datei *index.js* der Einstiegspunkt. webpack soll eine Bundle-Datei *bundle.js* im Verzeichnis *dist* des Projekts erzeugen. Diese Bundle-Datei referenziert man in der *index.html* (Listing 3).

Die Idee zu diesem Beispiel ist eine Website mit einem Artikel, an deren Ende sich ein Kommentarbereich (das Element *aside* mit der ID *comments)* befindet. Dieser Kommentarbereich soll mit React gerendert und der State mit Redux verwaltet werden.

Dumme Komponenten

Die Integration von React mit Redux teilt React-Komponenten in Smart Components und Dumb Components ein. Letztere sind schlicht und einfach herkömmliche React-Komponenten, die für sich nichts von einem vorhandenen Redux- ▶

Listing 2: webpack.config.js

```
var webpack = require('webpack');

module.exports = {
  devtool: "inline-sourcemap",
  context: __dirname + '/app',
  entry: "./index",
  output: {
    path: __dirname + "/dist",
    filename: "bundle.js"
  },

  module: {
    loaders: [
      {
        test: /\.js$/, exclude: [/node_modules/],
        loader: "babel-loader"
      }
    ]
  }
}
```

Container wissen. Die folgende Komponente soll Kommentarobjekte auf einer Website rendern:

```
import React from 'react'
let Comment = (props) => (
  <div className='comment'>{props.comment.text}</div>
);
```

Es wird ein Kommentarobjekt in den *props* erwartet, und dessen Attribut *text* enthält den Kommentartext. Es gibt keinerlei Verbindungen zu Redux. Beachten Sie hier, dass der Import von React notwendig ist, da die Komponente JSX-Code enthält, auch wenn es keine direkt sichtbare Verwendung der Variable React gibt.

Die zweite Komponente *NewCommentBox* ist zwar schon etwas komplizierter, aber auch diese ist eine Dumb Component (Listing 4). Über ein Textfeld kann man einen Kommentar eingeben, und bei einem Klick auf die Schaltfläche mit der Beschriftung *Save* werden der Wert des Textfelds und die aktuelle Zeit an den über die *props* übergebenen Event Handler *onSaveClick* übergeben. Die *NewCommentBox* wird mit einer Liste von Kommentaren zusammen als neue Komponente *Comments* zusammengefasst:

```
import React, {Component} from 'react';
import NewCommentBox from './NewCommentBox'
import Comment from './Comment'
export default (props) => {
  return (
    <div>
      <NewCommentBox
      onSaveClick={props.actions.createComment}/>
      <div>
        {
          props.comments.map(
            c=>(<Comment key={c.id} comment={c}/>)
          )
        }
      </div>
    </div>
  )
};
```

Die *NewCommentBox* bekommt hier einen Ereignis-Handler, der unter *props.actions.createComment* verfügbar sein soll. Die Liste der Kommentare wird einfach über die *props* in einem Attribut *comments* übergeben. Auch wenn dieser Code bereits ziemlich anwendungsspezifisch aussieht – auch Comments ist letztlich eine von Redux vollkommen unabhängige React-Komponente.

Reducer

Wie bereits beschrieben wird der in einem Redux-Store gespeicherte Zustand ausschließlich anhand von Actions und innerhalb der Reducer geändert. Der Reducer ist ein essenzieller Bestandteil des Systems und berechnet anhand des aktuellen Zustands und einer Action den Folgezustand. Für das Kommentar-UI kann der initiale Zustand und der zuständige Reducer wie in Listing 5 gezeigt aussehen.

Die Konstante *initialState* enthält den Zustand, mit dem die Anwendung initialisiert werden soll. Der Reducer selbst ist lediglich eine Funktion, die zwei Parameter erhält: *state* und

Listing 3: index.html

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8">
    <title>Ein Artikel</title>
  </head>
  <body>
    <h1>Artikel</h1>
    <p>
      Lorem ipsum dolor sit amet, consectetur
      adipisicing elit.
      Aspernatur blanditiis, doloribus libero porro
      quibusdamrecusandae voluptate voluptatum.
      Asperiores beatae deleniti, eius illum maxime,
      natus omnis
      quia quo, rerum sint voluptates.
    </p>
    <aside id="comments">

    </aside>
    <script src="dist/bundle.js"></script>
  </body>
</html>
```

Listing 4: NewCommentBox.js

```
import React, {Component} from 'react';

export class NewCommentBox extends Component {
  render() {
    return <div><input type="text" ref="input"/>
      <button onClick={(e)=> this.handleClick(e) }>
        Save
      </button>
    </div>
  }

  handleClick(ev) {
    const node = this.refs.input;
    const text = node.value.trim();
    this.props.onSaveClick(text, Date.now());
    node.value = ''
  }

}
```

action. In vielen Redux-Beispielen sieht man, dass Reducer mittels *switch* implementiert werden. Dafür gibt es keine Notwendigkeit. Es ist genauso leicht möglich, Objektliterale als Maps zu benutzen, um damit die Action-Typen auf Funktionen abzubilden.

Wie auch immer: Der Reducer ermittelt anhand des Action-Typs, was zu tun ist. Kennt der Reducer den Typ nicht, dann gibt er den übergebenen State unverändert zurück. Der initiale Zustand wird als Default-Parameter übergeben. Der Reducer des Beispiels kennt drei Actions:

- Mit *NEW_COMMENT* soll ein neuer Kommentar erzeugt werden können. Dazu übernimmt er den aktuellen Zustand und ersetzt lediglich das Array unter *comments*. Bei diesem übernimmt er wiederum alle bereits vorhandenen Elemente und fügt nur ein neues vorne an. Der Inhalt des neuen Kommentars wird aus den Attributen *text* und *id* der Action entnommen. Der ursprünglich übergebene Zustand bleibt unverändert. Dieses Beispiel benutzt die erst ab ES2016 angedachte *Object Spread*-Syntax, mit der es sehr leicht möglich ist, neue Objekte aus bestehenden zu erzeugen.
- Die Action *REQUEST_COMMENTS* wird dazu genutzt, zu signalisieren, dass die Anwendung Kommentare vom Server anfragt. Dies soll einmal nach dem Laden der Seite erfolgen. Der Zustand verändert sich lediglich in einem Punkt: Das Flag *requestingComments* wird auf *true* gesetzt, um anzuzeigen, dass gerade Daten geladen werden.
- Sobald die Daten vom Server empfangen wurden, kann die Action *RECEIVE_COMMENTS* genutzt werden, um die erhaltenen Kommentare verarbeiten zu lassen: Der Reducer ersetzt in diesem Fall einfach alle Kommentare mit den empfangenen und setzt auch das Flag *requestingComments* wieder auf *false*.

Mit diesem Reducer sind die Transformationsmöglichkeiten vollständig beschrieben. Es gibt keine Stelle im Programm, welche den Anwendungszustand unabhängig davon modifiziert. Wenn man den Code des Reducers näher betrachtet, fällt auf, dass es sich um reinen JavaScript-Code handelt. Zugegeben, es kommen einige über Babel bereitgestellte neue Features zum Einsatz, aber es gibt keinerlei Abhängigkeit zu einem Framework, auch nicht zu Redux. Ein Reducer ist schlicht und einfach eine JavaScript-Funktion mit einer definierten Schnittstelle.

Actions und Action Creators

Wie die Actions vom Reducer verarbeitet werden ist nun klar – doch wie werden sie erzeugt? Die Datei *actions.js* enthält die zugehörigen Action Creators:

```
// file: app/actions.js
export function createComment (text, id) {
```

▶

Listing 5: reducers.js

```
const initialState = {
  requestingComments: false,
  comments: []
};

export default function comments(state =
initialState, action) {
  switch (action.type) {
    case 'NEW_COMMENT':
      return {
        ...state,
        comments: [
          {text: action.text, id: action.id},
          ...state.comments
        ]
      };
    case 'REQUEST_COMMENTS':
      return {...state, requestingComments: true};
    case 'RECEIVE_COMMENTS':
      return {
        requestingComments: false,
        comments: action.payload
      };

    default:
      return state;
  }
}
```

Listing 6: Fortsetzung von actions.js

```
const storedComments = [
  { id:1, text: "Kommentar 1" },
  { id:2, text: "Kommentar 2" },
  { id:3, text: "Kommentar 3" }
];
export function requestComments () {
  return {
    type: "REQUEST_COMMENTS"
  }
}
export function receiveComments (comments) {
  return {
    type: "RECEIVE_COMMENTS",
    payload: comments
  }
}
export function fetchComments () {
  return (dispatch) => {
    dispatch(requestComments());
    setTimeout(()=>{
      dispatch(receiveComments
      (storedComments));
    }, 2000);
  }
}
```

```
  return {
    type: "NEW_COMMENT",
    text: text,
    id: id
  }
}
```

Der einfache Action Creator *createComment* erhält als Parameter den Text und die ID eines neuen Kommentars. Der Action Creator ist einfach nur eine Funktion, die ein Objektliteral mit dem Action-Typ und den notwendigen Daten der Action zurückgibt. Mehr ist nicht zu tun (Listing 6).

Die drei Action Creators *requestComments*, *receiveComments* und *fetchComments* sind ein zusammengehöriges Trio, das den Ablauf einer asynchronen Serverkommunikation beschreibt. Die ersten beiden Action Creators sind dabei noch herkömmlich und liefern einfach synchron die zugehörige Action zurück.

Bei *requestComments* genügt der Action-Typ, damit der Reducer weiß, dass er jetzt das Flag *requestingComments* setzen muss. Der Action Creator *receiveComments* ist ebenfalls synchron, er bekommt die empfangenen Kommentare als Parameter und erzeugt eine passende Action damit. Doch zwischen diesen beiden Actions findet üblicherweise eine asynchrone Kommunikation mit dem Server statt.

Redux kann jedoch im Basisumfang nur mit synchronen Action Creators umgehen. Für asynchrone Action Creators kann man Redux deshalb mit Zusatzmodulen erweitern.

Ein sehr einfaches Modul dafür ist redux-thunk, das vom selben Entwickler wie Redux stammt. Sobald diese Erweiterung aktiv ist, kann man in einem Action Creator nicht nur Actions (also Objekte mit Action-Typ-Attribut), sondern auch eine Funktion zurückgeben. Diese zurückgegebene Funktion erhält eine dispatch-Funktion als Parameter, mit der man eine Action an den Store schicken kann.

Der Action Creator *fetchComments* simuliert ein asynchrones Ablaufmuster durch einen Aufruf von *setTimeout*. Noch vor dem Aufruf wird mit *dispatch()* die Action *REQUEST_COMMENTS* per *requestComments()* erzeugt und an den Store geschickt. Danach erfolgt der Aufruf von *setTimeout()*. Wenn dieser abgelaufen ist, dann wird wiederum die Funktion *dispatch()* aufgerufen: Diesmal jedoch mit dem Ergebnis des Action Creators *receiveComments()*.

Links zum Thema

- Redux-Website
  http://redux.js.org
- Facebook Flux
  https://facebook.github.io/flux

Listing 7: App.js

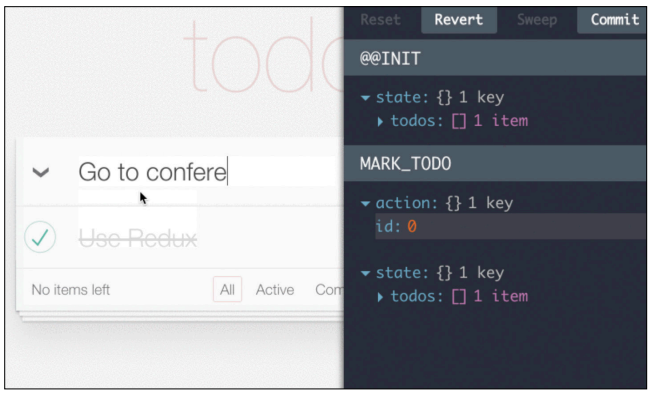
```
import React from 'react';
import {bindActionCreators} from 'redux'
import {Provider, connect} from 'react-redux'

import * as CommentActions from './actions'
import Comments from './components/Comments'

function mapStateToProps(state) {
  return {
    comments: state.comments,
    requestingComments: state.requestingComments
  }
}

function mapDispatchToProps(dispatch) {
  return {
    actions: bindActionCreators(CommentActions,
    dispatch)
  }
}

let App = (props) => (
  props.requestingComments
    ?<p>Requesting Comments...</p>
    :<Comments {...props}/>
);

export default connect(mapStateToProps,
mapDispatchToProps)(App);
```

Der Store und die Middlewares

Doch wie kann man eine Erweiterung wie redux-thunk zu Redux hinzufügen? Das Geheimnis sind die sogenannten Middlewares. Die Datei *store.js* enthält den Store des Beispielprogramms. Ein Redux-Store wird mit der Funktion

Listing 8: index.js

```
import React from 'react'
import {render} from 'react-dom'
import {Provider} from 'react-redux'

import store from './store'
import App from './App'
import {fetchComments} from './actions'

render(
  <Provider store={store}>
    <App comments={comments}/>
  </Provider>,
  document.getElementById("comments")
);

store.dispatch(fetchComments());
```



Werkzeuge wie die Redux-Devtools erleichtern das Debuggen (Bild 5)

createStore() erzeugt, indem man ihr die Reducer übergibt. Im einfachsten Fall reicht also:

```
// file: app/store.js (ohne Middleware)
import {createStore} from 'redux'
import reducers from './reducers'
export default createStore(reducers);
```

Beachten Sie: In dieser Datei wird überhaupt das erste Mal das Paket *redux* importiert. Alle bisherigen Module waren nicht abhängig von Redux, sondern bestehen letztlich aus herkömmlichem JavaScript. Diese geringe Kopplung macht eine Migration des Codes einfacher und zeigt, wie simpel Redux ist. Möchte man jedoch eine Redux-Erweiterung (Middleware) nutzen, dann benötigt man die Funktion *applyMiddleware* und erzeugt damit einen Wrapper um *createStore*:

```
import {createStore, applyMiddleware} from 'redux';
import thunk from 'redux-thunk';
import reducers from './reducers'
const createStoreWithMiddleware =
applyMiddleware(thunk)(createStore);
export default createStoreWithMiddleware(reducers);
```

Action Creators, Reducer und Store sind damit bereit. Auch die notwendigen Dumb-Components wurden bereits implementiert. Was nun noch fehlt, ist die Anbindung des Stores an diese React-Komponenten. Dazu dient eine sogenannte Smart-Component, die im Beispiel in der Datei *App.js* implementiert wird (Listing 7). Die Funktion *mapStateToProps()* erhält den aktuellen State vom Redux-Store und bildet ihn auf ein Objekt ab, das in die *props* der Smart Component eingemischt wird. Damit werden Inhalte des Redux-Store in React-Komponenten verfügbar gemacht. Damit die Dumb Components nichts vom Store und Dispatch wissen müssen, erzeugt man mit *bindActionCreators()* in der Funktion *mapDispatchToProps()* ein Objekt *actions*, das Callback-Funktionen für alle Actions aus *actions.js* enthält. Der Programmierer muss dann nur noch die Callback-Funktion aufrufen und kann so den Action Creator benutzen und per *dispatch()* die neue Action an den Store senden.

Die Komponente App erhält in ihren *props* die durch *mapStateToProps()* und *mapDispatchToProps()* bereitgestellten Daten und Action-Callbacks. Im Beispiel prüft die Komponente mit *props.requestingComments*, ob gerade Kommentare abgerufen werden. Ist dies der Fall, dann wird der Text *Requesting Comments…* angezeigt; andernfalls wird die Comments-Komponente erzeugt.

Die Verbindung zwischen dieser React-Komponente und dem Redux-Store erfolgt über den Aufruf der Funktion *connect* mit den Funktionen *mapStateToProps* und *mapDispatchToProps*. Diese Smart Component hängt nun wirklich von Redux ab und funktioniert nur, wenn eine ihrer Elternkomponenten einen Redux-Store anbietet. Wie dies funktioniert, sieht man in der Datei *index.js* (Listing 8).

Dies ist einerseits der Startpunkt der Anwendung, und andererseits sieht man hier auch den Toplevel-Render-Aufruf von React. Die Wurzelkomponente ist Provider aus dem Paket react-redux. Mit Provider stellt man einen Store an die Kinder der Komponente zur Verfügung – in diesem Fall ist das die Smart-Component-App.

Ein Toplevel-Dispatch von *fetchComments()* sorgt anschließend hier dafür, dass die Kommentare geladen werden und App sich entsprechend neu rendert.

Fazit

Redux setzt konsequent auf funktionale Konzepte, um ein erstaunlich kleines und dennoch mächtiges Framework zur Verwaltung des Anwendungszustands zu realisieren. Die Erweiterbarkeit durch Middlewares wird bereits für Debug-Werkzeuge, asynchrone Action Creators und vieles mehr genutzt. Entwickler, die noch wenig Erfahrung mit funktionaler Programmierung haben, tun sich naturgemäß schwerer mit dem Einstieg in ein solches Framework – dieser Aufwand lohnt sich jedoch. Es gibt kaum einen stärkeren Trend als den, funktionale Programmierkonzepte im Mainstream zu etablieren. Viele Entwickler erkennen heute, dass die Kombination aus veränderlichem Zustand (Mutable State) und Asynchronität zu schwer pflegbaren und instabilen Systemen führt.

Redux ist auch ein gutes Beispiel für den Entwurf moderner JavaScript-Frameworks: Actions, Action Creators, Reducers und State sind allesamt Plain Old JavaScript Objects. Das reduziert die Bindung an das Framework und erleichtert die Wiederverwendung des Codes bei einer Migration. Werkzeuge wie die Redux-Devtools (Bild 5) erleichtern das Debuggen erheblich. Selten kannte man vorher eine derart detaillierte Übersicht des Anwendungszustands. ■

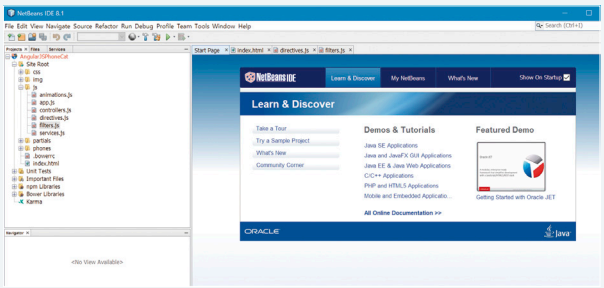
Jochen H. Schmidt
ist als Autor, Berater und Software-Entwickler tätig. Schwerpunkte seiner Aktivitäten sind Webentwicklung und Webtechnologien. Er ist Verfasser von bekannten Fachbüchern zum Thema Common Lisp.

ÜBERBLICK

CD-Highlights

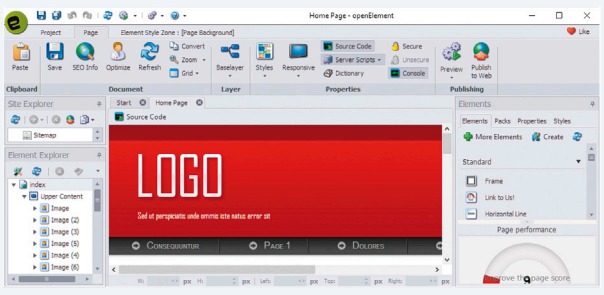
Auf der Heft-CD finden Sie Werkzeuge für Webentwickler sowie nützliche Tools.

NetBeans IDE 8.1



NetBeans IDE ist eine quelloffene Entwicklungsumgebung, die komplett in der Programmiersprache Java geschrieben wurde. NetBeans IDE unterstützt unter anderem C, C++ und dynamische Programmiersprachen. Zusätzlich wurden sogenannte Packs entwickelt, welche die IDE um Funktionsmöglichkeiten erweitern. Die IDE setzt ein Grundverständnis der verwendeten Programmiersprachen, Bibliotheken und Frameworks voraus. Das Paket enthält NetBeans Platform SDK, Java SE, Java FX, Java Web und EE, Java ME, Java Card 3 Connected, Ruby, C/C++, Groovy und PHP. Als Server werden GlassFish Server Open Source Edition und Apache Tomcat mitgeliefert.

openElement 1.50 R4



Das Web-Authoring-Programm unterstützt Sie beim Aufbau einer standardkonformen Website. Die Software nutzt dazu Technologien wie HTML5, CSS3 und jQuery. Sie können fertige Vorlagen einsetzen oder Projekte von Grund auf neu aufbauen. openElement wurde als visuelles Werkzeug entwickelt, um die Produktivität zu steigern. Der von openElement generierte Code ist zu 100 Prozent W3C-konform, SEO-ready und zeigt Webseiten korrekt auf jedem Browser oder Gerät.

Weitere Highlights:

Autohotkey 1.1.22.09

Mit kleinen Skripts automatisiert Autohotkey immer wiederkehrende Windows-Aufgaben. Für die tägliche Arbeit oder die Weitergabe per Download lassen sich die erstellten Skripts dank des mitgelieferten Compilers auch in eigenständige EXE-Programme umwandeln. Um neue Skripts zu erstellen, genügt ein einfacher Texteditor.

Ditto 3.21.28.0

Die Zwischenablage von Windows, die Sie zum Kopieren und Einfügen nutzen, bietet keine Extras. Das Gratis-Tool Ditto verwandelt die Zwischenablage in ein intelligentes Archiv, aus dem Sie jederzeit zuvor kopierte Bilder, Texte oder Dateien abrufen. Eine Suche fördert auch ältere Zwischenablageeinträge rasch wieder zutage.

Free File Sync 7.6

Das Tool erleichtert es, Daten an unterschiedlichen Speicherorten auf dem gleichen Stand zu halten. Dazu synchronisiert Free File Sync Ordner auf Festplatten, USB-Sticks, NAS-Servern und Online-Speichern. Dank eingebauter Versionskonfliktkontrolle werden keine Dateien versehentlich überschrieben.

WinSCP 5.7.6

Der grafische FTP-Client WinSCP sichert Ihre Datentransfers über das Internet per Secure FTP. Als Verschlüsselungsstandards stehen dazu SSH1 und SSH2 zur Verfügung. Oft genutzte Verbindungen lassen sich in einer Session-Liste speichern, und eine Synchronisationsoption gleicht das lokale und das entfernte Verzeichnis ab.

Windows System Control Center 2.5.0.3

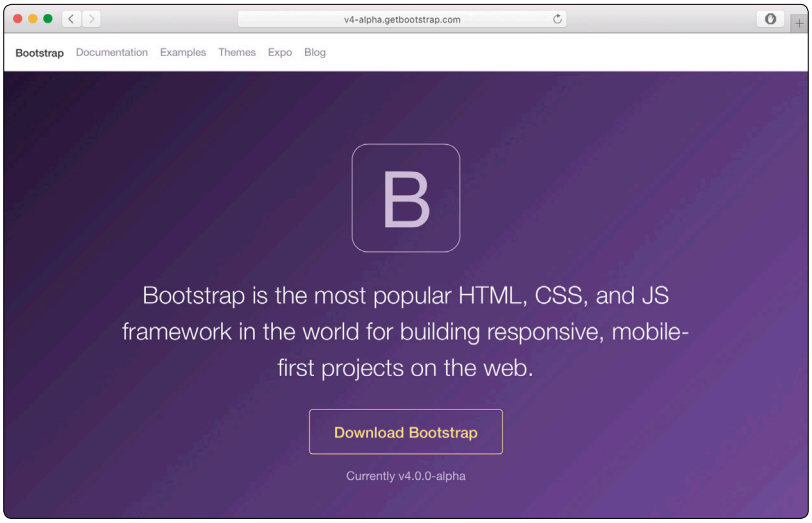
Mit dem WSCC als Schaltzentrale hat man von einem einheitlichen Menü aus Zugriff auf über 250 System-Tools. Ein Update-Manager hält die Hilfsprogramme stets aktuell. Installieren Sie das Programm und starten Sie es. Das Tool sucht nun online nach verfügbaren Programmen, lädt und installiert diese im Anschluss.

BOOTSTRAP 4

Versionssprung

Eineinhalb Jahre nach Bootstrap 3 setzt das Framework zum nächsten Versionssprung an.

Die Anforderungen an Frameworks für die Frontend-Entwicklung sind hoch: War früher der Hauptbestandteil solcher Frameworks CSS, so gehört heute selbstverständlich JavaScript für ausgefeilte interaktive UI-Komponenten wie etwa Tabs, Accordions und Dropdown-Menüs dazu. Zusätzlich muss ein Framework in der heutigen Zeit responsiv sein. Webentwickler erwarten außerdem oft ein ausgefeiltes Rastersystem, und schließlich sind viele auch an die komfortablere Arbeit mit CSS-Präprozessoren wie LESS und Sass gewöhnt, sodass das Framework auch hierfür eine Unterstützung bieten sollte. Ein Frameworks, das alle diese Erwartungen erfüllt, ist Boostrap.



Homepage: Die Bootstrap-Website (Bild 1)

Bootstrap wurde 2010 von Twitter (damals noch unter dem Namen Twitter Blueprint) zunächst für interne Zwecke entwickelt. Man suchte nach einer Lösung, um die Entwicklung von Webapplikationen auf Basis von HTML5 und CSS3 zu vereinfachen, um damit interne Analyse-Tools und Verwaltungswerkzeuge zu realisieren.

Bald schon merkte man aber, dass sich das Framework auch für andere Anwendungsfälle eignen würde, und hat es daher 2011 auf GitHub als Open-Source-Projekt freigegeben. Bootstrap bezeichnet sich selbst als das beliebteste HTML-, CSS- und JS-Framework der Welt, um responsive und Mobile-first-Projekte im Web zu erstellen. Der Claim ist zwar selbstbewusst, hat aber auch eine gewisse Substanz.

So ist das Framework beispielsweise seit Jahren eines der beliebtesten Repositories auf GitHub und man geht davon aus, dass 13 Prozent aller Websites, die JavaScript verwenden, auch Bootstrap haben. Das sind in etwa 9 Prozent aller Websites im Web. Damit ist Bootstrap nach jQuery die beliebteste JavaScript-Bibliothek überhaupt (Bild 1).

Das ist Grund genug für die Entwickler, diesen Erfolg zu festigen und das schon etwas betagte Framework einer Generalüberholung zu unterziehen.

Zum Zeitpunkt der Erstellung des Artikels befand sich das Framework noch in der Alpha-Phase, aber im Unterschied zu

Listing 1: index.html

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <!-- Required meta tags always come first -->
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width,
    initial-scale=1">
    <meta http-equiv="x-ua-compatible"
    content="ie=edge">

    <!-- Bootstrap CSS -->
    <link rel="stylesheet" href="css/bootstrap.css">
  </head>
  <body>

    <div class="container">
      <h1>Hello, world!</h1>
      <a href="#" class="btn btn-success">web & mobile
      DEVELOPER</a>
    </div>

    <!-- jQuery first, then Bootstrap JS. -->
    <script src="js/jquery.min.js"></script>
    <script src="js/bootstrap.js"></script>
  </body>
</html>
```

herkömmlicher Software kann man diesen Zustand bereits als sehr stabil bezeichnen. Es ist damit zu rechnen, dass die finale Version von Bootstrap 4 um den Jahreswechsel herum auf den Markt kommen wird.

Installation von Bootstrap

Man kann Bootstrap traditionell von http://getbootstrap.com/getting-started/#download herunterladen. Sollte dort die Version 4 noch nicht erhältlich sein, so können Sie auf http://v4-alpha.getbootstrap.com/getting-started/download ausweichen.

Grundsätzlich ist es entweder möglich, die Sourcen minifiziert und kompiliert zu laden oder aber komplett (inklusive Dokumentation, Sass-Dateien et cetera). Weiterhin ist es möglich, Bootstrap über einen der folgenden Paket-Manager zu erhalten:

- **Bower** *(bower install bootstrap)*
- **npm** *(npm install bootstrap)*
- **Meteor** *(meteor add twbs:bootstrap)*
- **Composer** *(composer require twbs/bootstrap)*

Zusätzlich gibt es ab sofort drei Custom-Builds, nämlich Reboot (enthält variables/mixins, Normalize und Reboot, aber kein JavaScript), Grid only (enthält variables/mixins und das Grid-System, aber kein JavaScript) und Flexbox (enthält Bootstrap mit aktiviertem Flexbox-Support).

Um die ersten Schritte mit Bootstrap zu machen, erstellen wir ein Verzeichnis *bootstrap* und dort zwei weitere Verzeichnisse *css* und *js*. In das Verzeichnis css kopieren wir die Datei bootstrap.css aus dem Unterverzeichnis *dist/css* der Bootstrap-Sourcen. Und in das *js*-Verzeichnis schließlich die Datei *bootstrap.js* aus dem Unterverzeichnis *dist/js*.

Nun benötigen wir noch jQuery, das wir unter https://ajax.googleapis.com/ajax/libs/jquery/2.1.4/jquery.min.js herunterladen und ebenfalls im *js*-Verzeichnis speichern können. Anschließend legen wir eine Datei *index.html* direkt im Verzeichnis *bootstrap* an (Listing 1).

Hier ist auch gut die Ladereihenfolge der einzelnen Komponenten zu sehen, die man einhalten sollte, damit Bootstrap einwandfrei funktioniert (Bild 2).

Beispiel: Das Hello World-Beispiel (Bild 2)

Bootstrap 4 unterstützt den IE 8 nun nicht mehr und kann daher mit einigen Features aufwarten (zum Beispiel *rem*-Unterstützung), die vorher nicht, nur schwierig oder nur mit Hilfe von Polyfills möglich waren.

Die Komponenten *well, panel* und *thumbnail* wurden entfernt und durch *cards* ersetzt. Glyphicons wurden komplett entfernt, da mit Font Awesome (https://fortawesome.github.io/Font-Awesome) sowie Octicons (https://octicons.github.com) leistungsfähige Alternativen zur Verfügung stehen.

Die primäre CSS-Einheit ist nun *rem* und nicht mehr *px*, die Einheit für die Media Queries *em* (anstelle von *px)*. Die globale Font-Größe ist von 14 px auf 16 px angewachsen. Für die Konfiguration des zu erzeugenden CSS gibt es nun die Datei *scss/_variables.scss*, in der man Einstellungen treffen und dann das CSS erneut kompilieren kann. Dazu zählen Farben, Optionen, Abstände, Einstellungen für das <*body*>-Element, Links, Grid Breakpoints, Grid Container, Grid Columns und Typografie.

Neue Utility-Klassen

Um Abstände und Ränder für beliebige Elemente festzulegen, wurde ein neues Set an Utility-Klassen eingeführt. Diese bestehen aus drei Teilen, die mit einem Minuszeichen getrennt sind. Margins werden mit *m*, Paddings mit *p* bezeichnet. Will man alle vier Seiten ansprechen, verwendet man *a*, ansonsten *t* für oben, *r* für rechts, *b* für unten *l* für links, x für rechts und links und y für oben und unten. Als dritte Kom- ▶

Listing 2: _utilities-spacing.scss

```
// Margins
.m-a-0 { margin:        0 !important; }
.m-t-0 { margin-top:    0 !important; }
.m-r-0 { margin-right:  0 !important; }
.m-b-0 { margin-bottom: 0 !important; }
.m-l-0 { margin-left:   0 !important; }
.m-x-0 { margin-right:  0 !important; margin-left:   0
!important; }
.m-y-0 { margin-top:    0 !important; margin-bottom: 0
!important; }

// $spacer steht für 1rem bzw. 16px
.m-a { margin:        $spacer !important; }
.m-t { margin-top:    $spacer-y !important; }

.m-a-md { margin:      ($spacer * 1.5) !important; }
.m-t-md { margin-top:  ($spacer-y * 1.5) !important; }

// Padding
.p-a-0 { padding:      0 !important; }
.p-t-0 { padding-top:  0 !important; }
```

ponente wird entweder *0* (für den Wert 0) oder aber der Breakpoint-Identifier angeben *(xs, sm* et cetera).

Listing 2 zeigt die zugehörigen Sass-Anweisungen zur besseren Erläuterung.

Sass statt LESS

Während Bootstrap 3 noch LESS als hauptsächlichen CSS-Präprozessor unterstützte (und ein Sass-Port erst nach dem Release von 3.0 auf den Markt kam), wurde dies in Bootstrap 4 zugunsten von Sass aufgegeben.

Die Gründe hierfür liegen auf der Hand: Einerseits konnte die Zeit für die Kompilierung dank *libSass* deutlich reduziert werden, und zudem ist die Entwicklergemeinde rund um Sass deutlich größer und aktiver, sodass zu erwarten ist, dass sich dies positiv auf das Projekt auswirken wird.

Überarbeitetes Grid

Das Grid-System wurde generalüberholt. Die Syntax wurde zwar nahezu beibehalten, intern werden aber jetzt für die Berechnung der Schriftgrößen *rem* (und nicht mehr *em)* verwendet. Damit ist nun responsive Typografie ernsthaft möglich.

Zudem wurde ein Breakpoint für besonders schmale Displays eingeführt (per Konfiguration unter 480 px). Dieser hat nun den Grid-Identifier *xs* erhalten, alle weiteren *(sm, md, lg, xl)* sind daher entsprechend breiter als früher. Und es gibt neue Mixins, um schnell Breakpoints zu erzeugen (Listing 3).

Flexbox

Obwohl der von Bootstrap 4 unterstützte Internet Explorer 9 Flexbox noch nicht unterstützt, wurde dennoch eine optionale Unterstützung dafür eingebaut. Diese muss allerdings in den Settings mit *$enable-flex: true !default;* eingeschaltet werden. Ist die Flexbox-Unterstützung aktiviert, werden folgende Änderungen vorgenommen:

- **Das gesamte Gridsystem** wird von *float* auf *display: flex;* umgestellt.
- **Input-Gruppen** werden von *display: table;* auf *display*: *flex;* umgestellt.
- **Die Media-Komponenten** werden ebenfalls von *display: table;* auf *display: flex;* umgestellt.

Bisher verwendete Bootstrap zum Reset *normalize.css* (https://necolas.github.io/normalize.css). Diese wurde nun durch die Eigenentwicklung *reboot.css* ersetzt, die auf Normalize basiert, aber nun deutlich eleganter und optimierter ist. Informationen hierzu gibt es unter http://v4-alpha.getbootstrap.com/content/reboot.

Card-Komponente

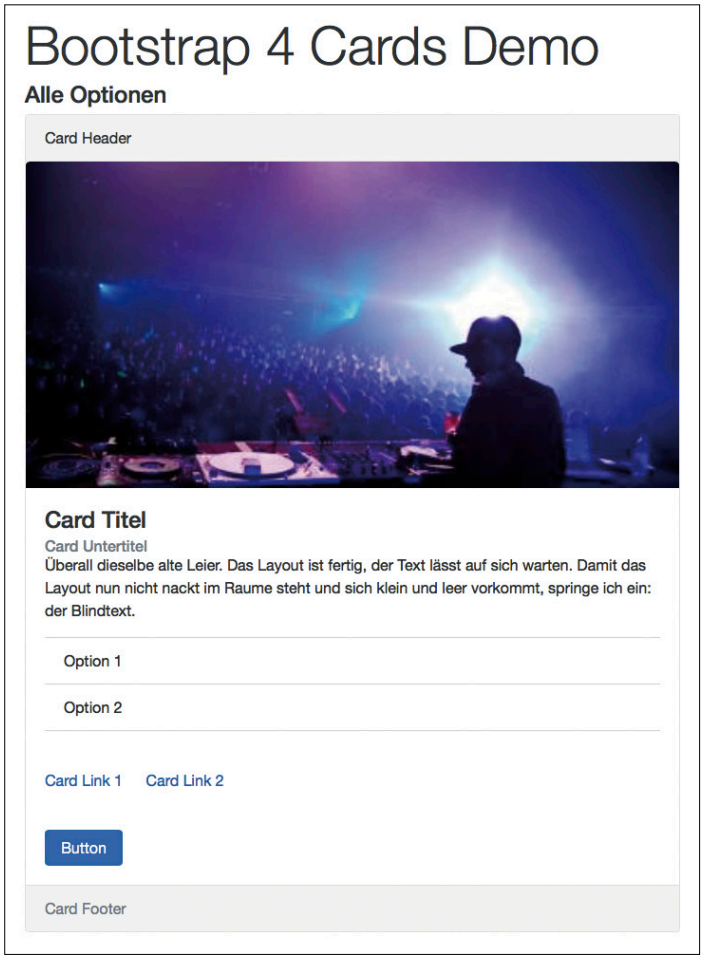
Die Komponenten Panels, Wells und Thumbnails wurden zugunsten der neuen Komponente Card aufgegeben. Diese konsolidiert die bisherigen Features, ist aber deutlich fle-

Listing 3: Mixins

```
// Erstellt eine Media Query: @media (min-width)
@include media-breakpoint-up(xs) { ... }
@include media-breakpoint-up(sm) { ... }
@include media-breakpoint-up(md) { ... }
@include media-breakpoint-up(lg) { ... }
@include media-breakpoint-up(xl) { ... }

// Erstellt eine Media Query: @media (max-width)
@include media-breakpoint-down(xs) { ... }
@include media-breakpoint-down(sm) { ... }
@include media-breakpoint-down(md) { ... }
@include media-breakpoint-down(lg) { ... }
@include media-breakpoint-down(xl) { ... }
```

Listing 4: Card – alle Optionen

```
<div class="container">
 <h3 class="display-1">Bootstrap 4 Cards Demo</h2>

 <h4 class="title">Alle Optionen</h4>
 <div class="demo-card">
  <div class="card">
   <div class="card-header">
    Card Header
   </div>
   <img class="card-img-top btn-block" src="http://
   lorempixel.com/400/200/"/>
   <div class="card-block">
    <h4 class="card-title">Card Titel</h4>
    <h6 class="card-subtitle text-muted">Card
    Untertitel</h6>
    <p class="card-text">Überall dieselbe alte
    Leier. Das Layout ist fertig, der Text lässt auf
    sich warten. Damit das Layout nun nicht nackt im
    Raume steht und sich klein und leer vorkommt,
    springe ich ein: der Blindtext.</p>
    <ul class="list-group list-group-flush">
     <li class="list-group-item">Option 1</li>
     <li class="list-group-item">Option 2</li>
     <li class="list-group-item">Option 3</li>
    </ul>
   </div>
   <div class="card-block">
    <a href="#" class="card-link">Card Link 1</a>
    <a href="#" class="card-link">Card Link 2</a>
   </div>
   <div class="card-block">
    <a href="#" class="btn btn-primary">Button</a>
   </div>
   <div class="card-footer text-muted">
    Card Footer
   </div>
  </div>
 </div>
</div>
```



Card Komponente mit allen Optionen (Bild 3)

Invertierte und eingefärbte Cards (Bild 4)

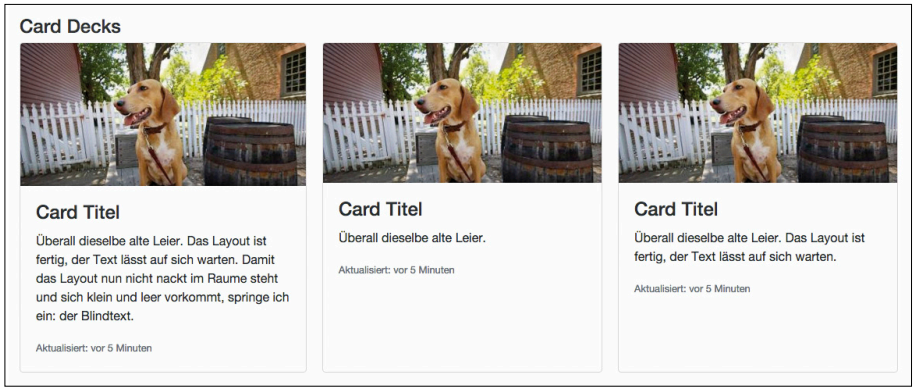
xibler. Schauen wir uns zunächst eine Card an, die möglichst viele Features beinhaltet (Listing 4).

Hier kann man gut den strukturellen Aufbau einer Card erkennen, die folgende Klassen bietet: *card* (Container, der die Card beinhaltet), *card-header* / *card-footer* (Header und Footer), *card-block* (Abschnitt in der Card), *card-text* (Text auf der Card), *card-img-top* / *card-img-bottom* (Bild oben beziehungsweise unten), *card-link* (Auszeichnung für Links) und *card-title* / *card-subtitle* (Titel und Untertitel) (Bild 3).

Die Komponente ist aber noch deutlich leistungsfähiger. So kann man beispielsweise die Card selbst invertieren und einfärben:

```
<div class="card card-inverse
card-primary text-center">
  <div class="card-block">
    <blockquote
    class="card-blockquote">
      <p>Das Layout ist fertig,
      der Text lässt auf sich
      warten.</p>
      <footer>Blindtext von <cite
      title="Source Title">
        http://www.blindtextgene
        rator.de</cite>
      </footer>
    </blockquote>
  </div>
</div>
```

Verwendet man die Klasse *card-primary*, erhält man eine blaue Card. Weitere mögliche Klassen an dieser Stelle sind: *card-success* (grün), *card-info* (hellblau), *card-warning* (gelb) und *card-danger* (rot) (Bild 4). Will man mehrere Cards nebeneinander anordnen (Bild 5), so kann man entweder Card ▶



Card Decks: Mehrere Cards nebeneinander anordnen (Bild 5)

Card mit Hintergrundbild (Bild 6)

Header 1	Header 2	Header 3
Zelle	Zelle	Zelle
Zelle	Zelle	Zelle
Footer 1	Footer 2	Footer 3

Invers: Tabellen sehr einfach invertieren (Bild 7)

Groups (dann gehören die Cards direkt zusammen) oder aber Card Decks (dann werden die Cards individuell behandelt) verwenden (Listing 5).

Will man zudem Masonry-Style Card Columns (http://masonry.desandro.com) aufbauen, verwendet man schlicht * * außen herum. Und schließlich kann man ein Bild auch als Hintergrund verwenden und den Text darüberlegen (Bild 6):

```
<article class="card card-inverse">
  <img class="btn-block img-responsive" src="http://
  lorempixel.com/400/200/" alt="Leaf on the street" />
  <div class="card-img-overlay">
    <h4 class="card-title">Card Titel</h4>
    <h6 class="text-muted">Autor</h6>
    <p class="card-text">Überall dieselbe alte Leier.
    Das Layout ist fertig, der Text lässt auf sich
    warten. Damit das Layout nun nicht nackt im Raume
    steht und sich klein und leer vorkommt, springe
    ich ein: der Blindtext.</p>
  </div>
</article>
```

In Bootstrap 4 ist es nun sehr einfach möglich, Tabellen zu invertieren – hierzu ist lediglich die Klasse *table-inverse* notwendig (Bild 7):

```
<table class="table table-inverse">
  <tr>
    <th>Footer 1</th>
    <th>Footer 2</th>
    <th>Footer 3</th>
  </tr>
</tfoot>
<tbody>
  <tr>
    <td>Zelle</td>
    <td>Zelle</td>
```

Listing 5: Card Decks

```
<div class="demo-card">
 <div class="card-deck-wrapper">
  <div class="card-deck">
   <div class="card">
    <img class="card-img-top img-responsive"
    src="http://lorempixel.com/400/200/">
    <div class="card-block">
     <h4 class="card-title">Card Titel</h4>
     <p class="card-text">Überall dieselbe alte
     Leier. Das Layout ist fertig, der Text lässt
     auf sich warten. Damit das Layout nun nicht
     nackt im Raume steht und sich klein und leer
     vorkommt, springe ich ein: der Blindtext.</p>
     <p class="card-text"> <small class="text-muted">
      Aktualisiert: vor 5 Minuten</small>
     </p>
    </div>
   </div>
   <div class="card">
    <img class="card-img-top img-responsive"
    src="http://lorempixel.com/400/200/">
    <div class="card-block">
     <h4 class="card-title">Card Titel</h4>
     <p class="card-text">Überall dieselbe alte Leier.
     </p>
     <p class="card-text"><small class="text-muted">
     Aktualisiert: vor 5 Minuten</small></p>
    </div>
    </div>
    <div class="card">
    <img class="card-img-top img-responsive"
    src="http://lorempixel.com/400/200/">
    <div class="card-block">
     <h4 class="card-title">Card Titel</h4>
     <p class="card-text">Überall dieselbe alte Leier.
     Das Layout ist fertig, der Text lässt auf sich
     warten. </p>
     <p class="card-text"><small class="text-muted">
     Aktualisiert: vor 5 Minuten</small></p>
    </div>
   </div>
  </div>
 </div>
</div>
```

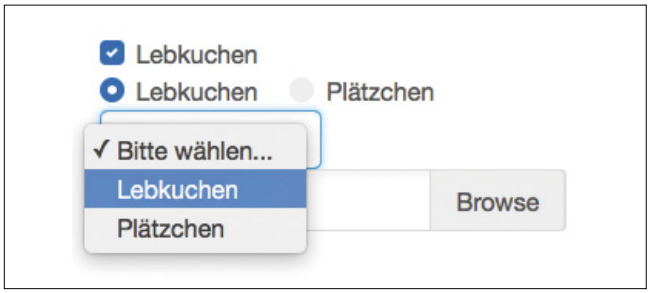
Listing 6: Eigene Formular-Elemente

```
<div>
  <label class="c-input c-checkbox">
    <input type="checkbox">
    <span class="c-indicator"></span>
    Lebkuchen
  </label>
</div>

<div>
  <label class="c-input c-radio">
    <input id="boots" name="radio" type="radio">
    <span class="c-indicator"></span>
    Lebkuchen
  </label>

  <label class="c-input c-radio">
    <input id="shoes" name="radio" type="radio">
    <span class="c-indicator"></span>
    Plätzchen
  </label>
</div>

<div>
  <select class="c-select">
    <option selected>Bitte wählen...</option>
    <option value="1">Lebkuchen</option>
    <option value="2">Plätzchen</option>
  </select>
</div>

<div>
  <label class="file">
    <input type="file" id="file">
    <span class="file-custom"></span>
  </label>
</div>
```

```
    <td>Zelle</td>
  </tr>
</tbody>
</table>
```

Zudem gibt es nun sogenannte Reflow-Tables (hier heißt die Klasse *.table-reflow)*. Damit werden die Spalten und Zeilen vertauscht. Weiterhin gibt es die folgenden kontextabhängigen Klassen: *.table-active, .table-success, .table-info, .table-warning* und *.table-danger*. Damit ist es möglich die Tabelle, Zeilen oder Felder einzufärben. Schließlich wurden Klassen eingeführt, um den Header zu invertieren *(.thead-inverse)* und einzufärben *(.thead-default)*.

Bootstrap 4 enthält darüber hinaus sogenannte Custom Forms, mit denen komplett eigene Formular-Elemente erzeugt werden, die einerseits in allen Browsern gleich aussehen sollen und zudem deutlich flexibler im Styling sind (Bild 8). Die originalen Formularelemente werden ausgeblendet, wie in Listing 6 dargestellt.



Eigene Formularelemente: Deutlich flexibler im Styling (Bild 8)

Fazit

Bootstrap hat seine Hausaufgaben gemacht und das ohnehin schon sehr gute Framework deutlich überarbeitet und modernisiert. Vor allem die Card-Komponente, aber auch die anderen Bestandteile und Änderungen sind gelungen und machen richtig Spaß in der Anwendung. ■

Links zum Thema

- Dokumentation zu Bootstrap 4 alpha
  http://v4-alpha.getbootstrap.com/getting-started/introduction
- Card-Demo Codepen
  http://codepen.io/alexdevero/pen/JYpMEO
- Unterschiede Bootstrap 3 und 4
  www.quackit.com/bootstrap/bootstrap_4/differences_between_bootstrap_3_and_bootstrap_4.cfm
- Bootstrap-4-Tutorials
  www.quackit.com/bootstrap/bootstrap_4/tutorial
- Bootstrap-4-Video-Tutorials
  https://www.youtube.com/watch?v=buzksTcGxZo

Patrick Lobacher
ist Digital-Native, Entwickler, Berater, Trainer, Coach und Autor zahlreicher Fachbücher und Fachartikel. Er ist Vorstandsvorsitzender der Pluswerk AG, die an zehn Standorten mit über 150 Mitarbeitern digitale Kommunikationslösungen konzipiert, umsetzt und betreut.

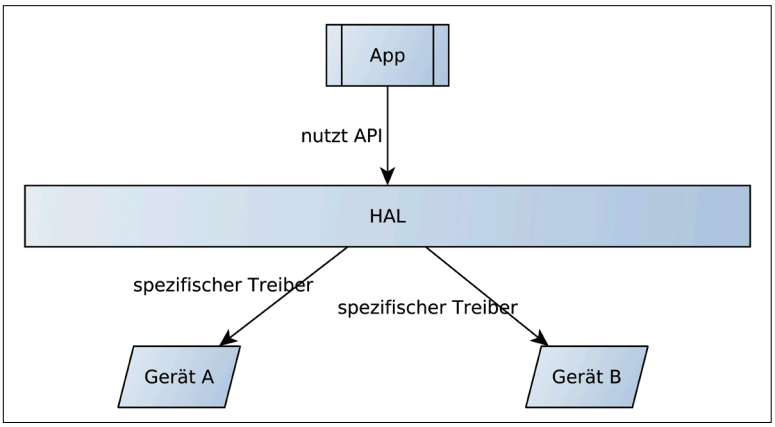
BLUETOOTH LE

Offener Industriestandard

Bluetooth LE ist dank seines geringen Stromverbrauchs für das Internet of Things von eminenter Bedeutung.

Das von der Bluetooth SIG als offener Industriestandard verwaltete Funkprotokoll Bluetooth funktionierte problemlos, weil die diversen unterschiedlichen Peripheriegeräte über eine vergleichsweise kleine Gruppe von Profilen abgedeckt waren (Tabelle 1).

Interessant wurde die Situation durch das Aufkommen preiswerter Bluetooth-Module, die das Einbauen von Funk in Alltagsgagdets und Whiteware ermöglichten. Da die Entwicklung und Spezifizierung eines neuen Profils immens aufwendig ist, nutzten die Hersteller stattdessen proprietäre Funkschnittstellen auf Basis des seriellen Kommunikationsprotokolls SPP.



HALs trennen Logik und Treiber voneinander (Bild 1)

Für App-Entwickler ergab dies eine höchst unbefriedigende Situation: Wer in seinem Programm Hardware von zwei verschiedenen Anbietern unterstützen möchte, musste eine Hardware-Abstraktionsschicht implementieren (Bild 1).

Zudem erwies sich das verbindungsorientierte Funkprotokoll für das Internet of Things (IoT) als weniger gut geeignet. Der Scanprozess, der immerhin rund vierzehn Sekunden dauert, ist in der Praxis ärgerlich. Außerdem war der Stromverbrauch alles andere als gering.

Unterschiede zum klassischen Bluetooth

Bluetooth LE unterscheidet sich aus physikalischer Sicht nur wenig von klassischem Bluetooth. Das charakteristische Frequenz-Hopping ist abermals mit von der Partie. Dadurch ist sichergestellt, dass das Funkprotokoll auch in belasteten Umgebungen problemlos funktioniert.

Auf logischer Ebene gibt es massive Differenzen. Bluetooth LE basiert im Grunde genommen auf den zwei in Bild 2 gezeigten Profilen, die gemeinsam den Gutteil aller Einsatzszenarien abdecken.

GAP – der Begriff steht für Generic Access Profile – ist für die Präsenzerkennung zuständig. Das Protokoll unterteilt Geräte in Peripherals und Centrals. Erstere sind Datenquellen, während die auf die Informationen zugreifenden Telefone oder Computer als Central bezeichnet werden.

GAP-basierte Geräte kündigen ihre Anwesenheit durch das Versenden von bis zu 31 Byte langen Paketen an, die als Advertising Data bezeichnet werden. Ein optionaler zweiter Teil der Spezifikation befasst sich mit dem Entgegennehmen von Scan Response Requests, die mit einem ebenfalls bis zu 31 Byte langen Scan Response Data-Paket beantwortet werden (Bild 3).

Der eigentliche Austausch der Daten erfolgt über das als GATT bezeichnete Generic Attribute Profile. Es spezifiziert Möglichkeiten zum Austausch von Characteristics, die vom Aufbau her an KV-Speicher erinnern. Zum Zweck der einfacheren Katalogisierung spezifiziert die Bluetooth SIG Profile und Services. Bild 4 zeigt, wo die einzelnen Elemente von GATT im Gesamtsystem einzuordnen sind.

Deskriptoren dienen dabei – analog zu den in OpenStreetMap und in Graphendatenbanken verwendeten Tags – als Ablage für zusätzliche Informationen. Ihre Verwendung lohnt sich, wenn der Stromverbrauch auf Seiten des Senders minimiert werden soll: Wer eine geringere Datenmenge überträgt, verbraucht auch weniger Strom.

Serverspiele

Da der im Android SDK enthaltene Emulator Bluetooth nicht unterstützt, wollen wir mit der Entwicklung eines GATT-Peripheriegeräts beginnen. Das dazu notwendige API steht erst ab Android 5.0 zur Verfügung. Erstellen Sie in Android Studio ein Projektskelett namens *NMGGattServer* und wählen Sie als Minimum-SDK die Version 5.0 aus.

Zum Zugriff auf die Bluetooth-Hardware sind Permissions erforderlich. Öffnen Sie die Manifestdatei und adaptieren Sie den Permissions-Block nach folgendem Schema:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android=
http://schemas.android.com/apk/res/android
package="com.tamoggemon.nmggattserver" >
<uses-permission android:name=
"android.permission.BLUETOOTH" />
```

```
<uses-permission android:name=
"android.permission.BLUETOOTH_ADMIN" />
...
```

Google unterteilt die für Bluetooth zuständigen Permissions: *BLUETOOTH* erlaubt das Verbinden mit dem Gerät bekannten Partnern, während die Suche nach neuen Gegenstellen die nur im Zusammenspiel mit *BLUETOOTH* funktionierende Permission *BLUETOOTH_ADMIN* voraussetzt.

Der im Play Store enthaltene Zuteilungsalgorithmus kann Ihre Applikation auf Geräte beschränken, die mit einem Bluetooth-LE-Radio ausgestattet sind. Dazu müssen Sie folgende Passage in die Manifestdatei einbinden:

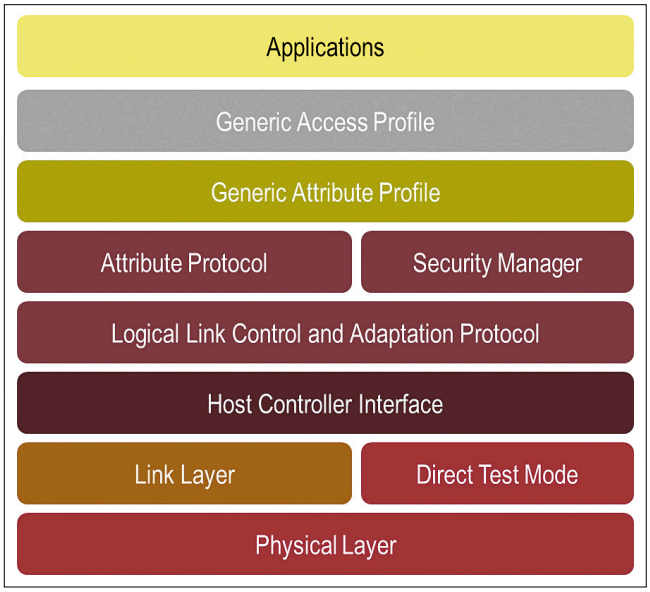
```
<manifest xmlns:android=
http://schemas.android.com/apk/res/android
package="com.tamoggemon.nmggattserver" >
<uses-feature android:name=
"android.hardware.bluetooth_le"
android:required="true"/>
```

Google bittet in der Dokumentation darum, dass der *uses*-Block immer eingebunden wird. Wenn Ihr Produkt auch ohne Bluetooth-LE-Radio funktioniert, nutzen Sie stattdessen – wie in unserem Beispiel – folgende Syntax:

```
<uses-feature android:name=
"android.hardware.bluetooth_le"
android:required="false"/>
```

Nach dem Laden der Activity prüft unser Programm, ob ihm Bluetooth-Hardware zur Verfügung steht. Dies erfolgt durch folgenden Code, der die Activity auf nicht kompatiblen Telefonen durch Aufruf von *finish()* beendet:

```
protected void onCreate(Bundle savedInstanceState) {
  ...
  if (!getPackageManager().hasSystemFeature(
  PackageManager.FEATURE_BLUETOOTH_LE)) {
    Toast.makeText(this, "Dieses Telefon unterstützt
    BTLE nicht", Toast.LENGTH_SHORT).show();
    finish();
  }
}
```



Der Bluetooth-LE-Stack ist vergleichsweise einfach (Bild 2)

Im nächsten Schritt müssen wir uns mit dem Bluetooth-LE-Stack des Telefons verbinden. Die Managerklasse dient dabei als Schnittstelle zum Stack, während der eigentliche Transmitter über eine Instanz von BluetoothAdapter abgebildet wird:

```
public class MainActivity extends Activity {
  BluetoothAdapter myAdapter;
  BluetoothManager myManager;
  @Override
  protected void onCreate(Bundle savedInstanceState)
  {
    ...
    BluetoothManager myManager;
    myManager = (BluetoothManager)
    getSystemService
    (Context.BLUETOOTH_SERVICE);
    myAdapter = myManager.getAdapter();
    if (myAdapter == null) {
      Toast.makeText(this,
      "Adaptererstellung fehlgeschlagen",
      Toast.LENGTH_SHORT).show();
      finish();
      return;
    }
```

Zur Erstellung eines Bluetooth-LE-Servers rufen wir *openGattServer* auf und schreiben sogleich einen Service samt dazugehörender Characteristic ein: ▶

Tabelle 1: Die Bluetooth SIG spezifiziert diverse Profile

Profil	Kurzbeschreibung
Advanced Audio Distribution Profile	Audioübertragung per Bluetooth-Luftschnittstelle
Dial-up Networking Profile	Simuliert Modemverbindungen
Generic Access Profile	Grundlegendes Protokoll, das für Gerätefindung zuständig ist
LAN Access Profile / Personal Area Networking Profile	Protokollfamilie, die Bluetooth zur Errichtung eines Netzwerks für höhere Protokolle nutzt.
Object Push Profile	An OBEX angelehntes Protokoll zum Austausch von generischen Dateien
Serial Port Profile	Simuliert RS232-artige Verbindung über die Funkschnittstelle
SIM Access Profile	Teilt die in einem Telefon befindliche SIM-Karte mit anderen Geräten (Stichwort Autotelefon)

```
TextView myTextView=(TextView)findViewById(R.id.
textView);
myServer = myManager.openGattServer(this,
new NMGGattServerObserver(this, myTextView));
BluetoothGattService s = new BluetoothGattService
(NMGConstants.ServiceUUID,
BluetoothGattService.SERVICE_TYPE_PRIMARY);

myCharacteristic = new BluetoothGattCharacteristic
(NMGConstants.CharacteristicUUID,
BluetoothGattCharacteristic.PROPERTY_READ,
BluetoothGattCharacteristic.PERMISSION_READ );
myCharacteristic.setValue("NMG Test");
s.addCharacteristic(myCharacteristic);
myServer.addService(s);
```

Im Rahmen der Erstellung des *BluetoothGattCharacteristic*-Objekts fragt der Bluetooth-Stack die Eigenschaften der Characteristic ab. Der Konstruktor nimmt zwei Bitfelder entgegen: Feld eins beschreibt die Eigenschaften der neu zu erstellenden Characteristic, während Feld zwei die Lese- beziehungsweise Schreibberechtigungen beschreibt.

Services und die zu ihnen gehörenden Charakteristika werden durch GUIDs gekennzeichnet. Es handelt sich dabei um eine sehr lange Zeichenfolge, die nach einem bestimmten Schema aufgebaut ist. Ob der enormen Länge sind Kollisionen sehr unwahrscheinlich. Beschaffen Sie sich die benötigten GUIDs einfach aus einem GUID-Generator Ihrer Wahl. Im Fall unseres Programmbeispiels liegen die GUIDs zwecks einfacherem Handling in einer Convenience-Klasse, die sie als zwei statische Variablen exponiert:

```
public class NMGConstants {
  public static UUID
  CharacteristicUUID;
  public static UUID ServiceUUID;
  static{
    CharacteristicUUID=
    UUID.fromString
    ("0acea172-2a76-69a7-4e49-
    302ed371f6a8");
    ServiceUUID=UUID.fromString
    ("abcbe138-a00c-6b8e-7d44-
    4b63a80170c3");
  }
}
```

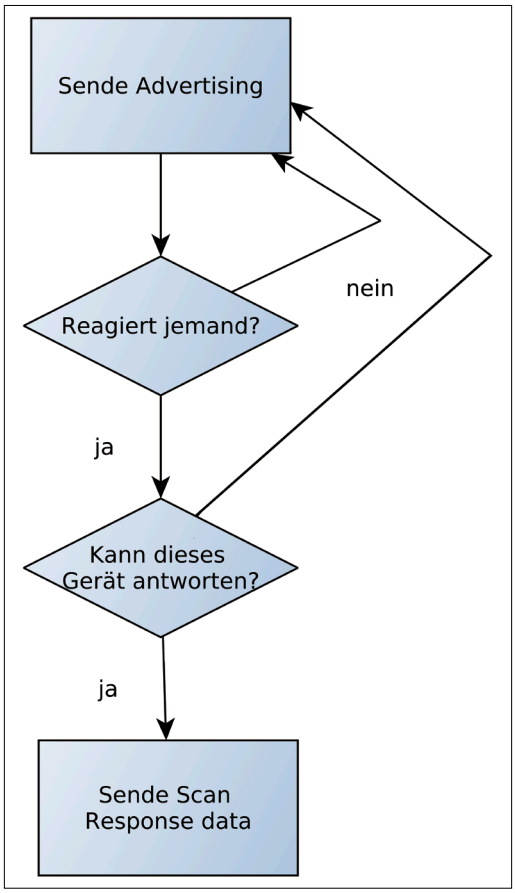
Damit fehlt nur noch die Einrichtung des Advertisers. Diese Klasse ist für das Versenden der GAP-Präsenzinformationen erforderlich: Clients können den Server nur dann finden, wenn der Advertiser aktiv ist. Im Fall unseres Programms beginnt die Kommandofolge mit dem Beschaffen einer Advertiserklasse, die sodann gegen null geprüft wird:

```
myAdvertiser = myAdapter.getBluetoothLeAdvertiser();
if(myAdvertiser==null){
  Toast.makeText(this, "Servermodus nicht erlaubt!",
  Toast.LENGTH_SHORT).show();
  finish();
}
```

Google schaltet die Bluetooth-LE-Serverfähigkeit nur auf jenen Telefonen frei, deren Hardware gleichzeitig als Server und als Client agieren kann. Bei Nur-Clients liefert *getBluetoothLeAdvertiser* den Wert *null* zurück. Unter http://altbeacon.github.io/android-beacon-library/beacon-transmitter-devices.html findet sich eine von Drittanbietern gepflegte Kompatibilitätsliste, die weitere Informationen über Hardwaresupport anbietet.

Wenn das als Basis dienende Gerät als Advertiser dienen kann, so folgt die eigentliche Aktivierung der Klasse. Neben Einstellungen zur zu verwendenden Sendeenergie sind auch Informationen über die zu bewerbenden Dienste erforderlich:

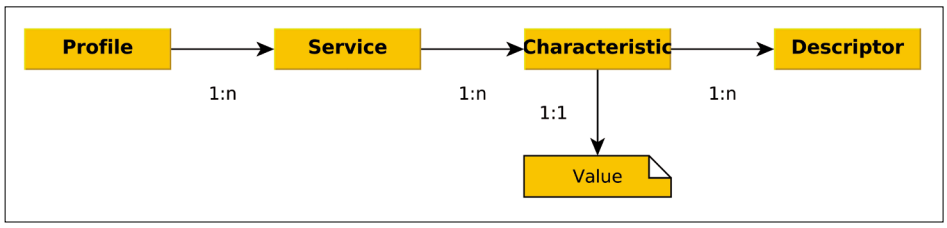
```
else {
  AdvertiseSettings advertiseSettings = new
                                AdvertiseSettings.Builder()
    .setTxPowerLevel
    (AdvertiseSettings.
    ADVERTISE_TX_POWER_MEDIUM)
    .setConnectable(true)
    .setAdvertiseMode
    (AdvertiseSettings.
    ADVERTISE_MODE_BALANCED)
    .build();
  AdvertiseData advertiseData =
  new AdvertiseData.Builder()
    .setIncludeTxPowerLevel(false)
    .addServiceUuid(new ParcelUuid
    (NMGConstants.ServiceUUID))
    .setIncludeDeviceName(true)
    .build();
  myAdvertiser.startAdvertising
  (advertiseSettings,
  advertiseData, advertiseData,
  advertiseCallback);
```



Der GAP-Prozess wird im per Advertising Interval vorgegebenen Takt abgearbeitet (Bild 3)

Wie im Fall des Servers ist auch hier ein Observer erforderlich, der vom Betriebssystem über eingehende Ereignisse informiert wird. Im Fall des Advertisers reicht es aus, eine Instanz der vom System vorgegebenen Klasse zu verwenden. Achten Sie darauf, dass das Aktivieren des Scanmodus zu einem massiven Anstieg des Energieverbrauchs führt.

Wir wollen an dieser Stelle ein Textfeld in die Hauptseite einpflegen, das im Lauf der Programmausführung mit weiteren Informationen über die anfallenden Ereignisse bevölkert wird. Diese Vorgehensweise ist – unter anderem – deshalb hilfreich, weil die Aktivitäten des Servers auf diese Weise auch ohne ADB-Konsole analysiert werden können:



GATT definiert eine Gruppe von Elementen, die untereinander in 1:n-Beziehungen stehen (Bild 4)

```
<TextView android:text="@string/hello_world"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent"
  android:id="@+id/textView" />
```

Bluetooth-Operationen können langwierig sein. Zwecks Sicherung der Reaktivität müssen Aufrufer ein Callback-Objekt übergeben, dessen Methoden vom Bluetooth-Stack beim Auftreten bestimmter Ereignisse aufgerufen werden.

Alles asynchron

NMGGattServerObserver ist eine Implementierung des in Listing 1 näher beschriebenen Observers. Wir beschränken uns im Moment auf das Ausgeben von Statusinformationen. Dank des Aufrufs der in der Superklasse implementierten Kommunikationslogik müssen wir selbst keine Kommandos an den GATT-Server absetzen.

Im Rahmen der Erzeugung der Listener-Klasse schreiben wir die *TextView*-Instanz in eine globale Variable, in der sie später zur Verfügung steht. Da Google *BluetoothGattServerCallback* mitunter außerhalb des GUI-Threads aufruft, implementieren wir zudem eine generische Methode zur sicheren Manipulation des Inhalts der Textbox:

```
public class NMGGattServerObserver extends
BluetoothGattServerCallback {
  MainActivity myActivity;
  TextView myView;
  public void updateView(final String _s) {
    new Handler(Looper.getMainLooper()).post(new
    Runnable() {
      @Override
      public void run() {
      myView.setText(myView.getText() + "\n" + _s); }
    });
  }
  public NMGGattServerObserver(MainActivity _a,
  TextView _v) {
    super();
    myActivity =_a;
    myView=_v;
  }
```

Damit können wir uns der eigentlichen Kommunikation zuwenden. Verbindungen und Trennungen werden vom Betriebssystem durch Aufrufe von *onConnectionStateChange* angezeigt. Unsere App quittiert sie mit dem Absetzen einer Meldung in die Textbox:

```
@Override
public void onConnectionStateChange(BluetoothDevice
device, int status, int newState) {
  if(newState== BluetoothProfile.STATE_CONNECTED)
  {
    updateView("Verbindungszustand geändert:
    VERBUNDEN");
  }
  else if(newState==BluetoothProfile.STATE_DISCONNECTED)
  {
    updateView("Verbindungszustand geändert: GETRENNT");
  }
  super.onConnectionStateChange(device, status,
  newState);
}
```

Eingehende Leseereignisse handelt unser Listener nach demselben Schema ab. Der Aufruf von *super()* sorgt dafür, dass die in Android enthaltene Kommunikationslogik mit der Verarbeitung des Ereignisses betraut wird:

```
@Override
public void onCharacteristicReadRequest(BluetoothDevice
device, int requestId, int offset, BluetoothGatt
Characteristic characteristic) {
  updateView("Leserequest ist eingegangen");
  super.onCharacteristicReadRequest(device, requestId,
  offset, characteristic);
  myActivity.myServer.sendResponse(device, requestId,
  BluetoothGatt.GATT_SUCCESS, offset,
  characteristic.getValue());
}
```

Bedauerlicherweise bietet Android – zumindest zum Zeitpunkt der Drucklegung – keine Logik an, die das Senden der in der Charakteristik befindlichen Informationen automatisiert. Das Übertragen der Daten muss auf jeden Fall durch Aufruf von *sendResponse* bewerkstelligt werden. Unterbleibt dies, so erhält der Client keine Daten zurück.

Im Fall von Android-Clients wäre dieses Fehlverhalten besonders kritisch, weil das Betriebssystem in einer Endlosschleife auf das Eingehen einer Antwort wartet und den *onCharacteristicRead*-Callback nicht abfeuert. ▶

Starten Sie das System sodann auf Ihrem Zieltelefon. Wenn der Start der Applikation mit einer Fehlermeldung zu inkompatibler Hardware quittiert wird, so müssen Sie ein kompatibles Telefon erwerben. Die im Abschnitt zu Android for X86 besprochene VM ist auf die Rolle des Clients beschränkt. Erfreulicherweise ist das Motorola G der ersten Generation für vergleichsweise kleines Geld zu haben. Unsere App funktionierte auf einem Moto G mit Android 5.1 ohne jedes Problem.

Und jetzt auf dem Client

Damit können wir uns der Realisierung eines Clients zuwenden. Erstellen Sie ein zweites Projekt namens *NMGGattClient*, das die im vorigen Abschnitt beschriebenen Manifest-Anpassungen analog erhält. Es darf als Zielsystem Android 4.4 bekommen.

Das Client-API steht auch auf früheren Versionen des Betriebssystems zur Verfügung. Leider gibt es in Android 4.3 einige Probleme und Bugs, die unter http://stackoverflow.com/questions/17870189/android-4-3-bluetooth-low-energy-unstable?rq=1 im Zusammenspiel mit Methoden zur Umgehung präsentiert werden.

Das Programm enthält neben einer Liste der gefundenen Geräte auch zwei Buttons, die das Lesen und Schreiben der NMG-Charakteristik erlauben. *OnCreate* beginnt mit der Erzeugung der grundlegenden Bluetooth-Klassen und der hier aus Platzgründen nicht abgedruckten Verdrahtung von Buttons und Event Handlern:

```
@Override
protected void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  setContentView(R.layout.activity_main);
  myManager = (BluetoothManager)
  getSystemService(Context.BLUETOOTH_SERVICE);
  myAdapter = myManager.getAdapter();
  myHandler = new Handler();
  myGattCallback=new NMGGattCallback(this);
  ...
```

Für die Liste sind zwei globale Arrays erforderlich. Array Nummer 1 enthält die Namen der Geräte, während die dazugehörenden *BluetoothDevice*-Instanzen in einem separaten Array landen. Die Korrelation zwischen den beiden Feldern erfolgt über den Index nach dem Schema eins zu eins:

```
public class MainActivity extends AppCompatActivity
implements View.OnClickListener {
  public ArrayAdapter<String> listAdapter ;
  public LinkedList<BluetoothDevice> myList;
  ...
  @Override
  protected void onCreate(Bundle savedInstanceState) {
    ...
    listView = (ListView) findViewById( R.id.listView );
    ArrayList<String> aList = new ArrayList<String>();
    listAdapter = new ArrayAdapter<String>(this,
    R.layout.nmgrow, aList);
    listView.setAdapter( listAdapter );
    myList=new LinkedList<BluetoothDevice>();
```

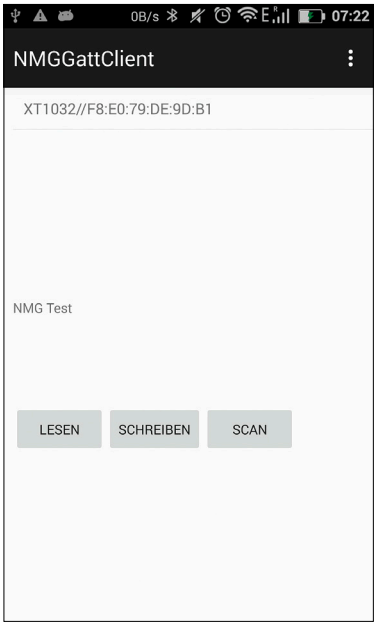
Android-Puristen wenden an dieser Stelle ein, dass die Verwendung eines von Hand geschriebenen Adapters den Vorgaben von Googles Dokumentation eher entspricht. Für uns ist dies insofern irrelevant, als wir Elemente nicht dynamisch hinzufügen beziehungsweise entfernen wollen.

Suche nach einem anzusprechenden Server

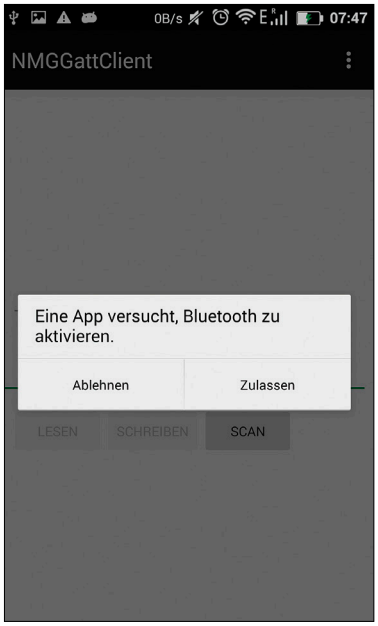
Im nächsten Schritt müssen wir die Suche nach einem anzusprechenden Server anstoßen. Hierzu ist folgender Code erforderlich:

```
@Override
protected void onCreate(Bundle
savedInstanceState) {
  ...
  myScanCallback=new NMGScanCallback(this);
  scanDevices();
}
private void scanDevices()
{
  myHandler.postDelayed(new Runnable() {
    @Override
    public void run() {
      myAdapter.stopLeScan(myScanCallback);
    }
  }, 5000); //5 sec
  myAdapter.startLeScan(myScanCallback);
}
```

Die Suche nach Bluetooth-Hardware ist kein stromsparender Prozess. Es ist im Interesse der



NMGGattClient hat die Charakteristik erfolgreich ausgelesen (Bild 5)



Aktivierung: Bitte aktivieren Sie Ihren BT-Transmitter (Bild 6)

Listing 1: GattServerCallback

```
public abstract class BluetoothGattServerCallback {
  public void onConnectionStateChange
  (BluetoothDevice device, int status,
    int newState) {}
  public void onServiceAdded(int status,
  BluetoothGattService service) {}
  public void onCharacteristicReadRequest
  (BluetoothDevice device, int requestId,
    int offset, BluetoothGattCharacteristic
    characteristic) {}
  public void onCharacteristicWriteRequest
  (BluetoothDevice device, int requestId,
    BluetoothGattCharacteristic characteristic,
    boolean preparedWrite, boolean
    responseNeeded, int offset, byte[] value) {}
  public void onDescriptorReadRequest
  (BluetoothDevice device, int requestId,
    int offset, BluetoothGattDescriptor
    descriptor) {}
  public void onDescriptorWriteRequest
  (BluetoothDevice device, int requestId,
    BluetoothGattDescriptor descriptor,
    boolean preparedWrite,
    boolean responseNeeded, int offset,
    byte[] value) {}
  public void onExecuteWrite(BluetoothDevice device,
  int requestId, boolean execute) {}
  public void onNotificationSent(BluetoothDevice
  device, int status) {}
  public void onMtuChanged(BluetoothDevice
  device, int mtu) {}
}
```

Vermeidung von Pollution sinnvoll, das Suchintervall so kurz wie möglich zu halten. Martin Woolley – der Brite arbeitet bei der Bluetooth SIG – zeigt bei Präsentationen Scandauern im Bereich von einer Sekunde und bezeichnet diese als konservativ. Unser Wert von fünf Sekunden ist also mehr als defensiv.

Da der Bluetooth-Stack asynchron aufgebaut ist, müssen wir die Entgegennahme der Ergebnisse in Form einer Listener-Klasse aufbauen. *NMGScanCallback* sieht so aus:

```
public class NMGScanCallback implements
BluetoothAdapter.LeScanCallback {
  MainActivity myAct;
  NMGScanCallback(MainActivity _act){
    super();
    myAct=_act;
  }
  @Override
  public void onLeScan(BluetoothDevice device, int rssi,
  byte[] scanRecord){
    final BluetoothDevice device1=device;
    myAct.runOnUiThread(new Runnable() {
      @Override
      public void run() {
        myAct.listAdapter.add(device1.getName() + "//" +
        device1.getAddress());
        myAct.listAdapter.notifyDataSetChanged();
        myAct.myList.add(device1);
      }
    });
  }
}
```

onLeScan wird beim Finden eines neuen Geräts aufgerufen. Wir speichern die zurückgegebenen Ergebnisse im von der Liste verwendeten Array – der in *rssi* angelieferte Wert für die Signalstärke wird hier nicht weiterverarbeitet.

Beim Anklicken eines Elements der Liste kommt folgender Code zum Einsatz:

```
listView.setOnItemClickListener(new
AdapterView.OnItemClickListener() {
  @Override
  public void onItemClick(AdapterView<?> parent,
  View view,
    int position, long id)
  {
    myGattConnection=myList.get(position).connectGatt
    (getApplicationContext(), false, myGattCallback);
    myGattConnection.discoverServices();
  }
});
```

Beim Erzeugen einer Instanz der *GattConnection*-Klasse ist ein Verweis auf ein *GattCallback* zwingend erforderlich. Er wird vom System über Änderungen im Verlauf des Suchprozesses informiert. Die im Konstruktor notwendige Zuweisung der *MainActivity*-Instanz wird hier aus Platzgründen nicht abgedruckt.

Im Fall eines erfolgreichen Verbindungsaufbaus zum Endgerät wird eine Meldung in die Debuggerkonsole ausgeworfen. Zudem folgt ein Aufruf von *discoverServices*, der für das Auflisten aller auf dem Gerät implementierten Charakteristika zuständig ist:

```
@Override
public void onConnectionStateChange(BluetoothGatt
gatt, int status, int newState) {
  super.onConnectionStateChange(gatt, status, newState);
  if (newState == BluetoothProfile.STATE_CONNECTED) {
    Log.i("NMG", "Beginne Suche:" +
    gatt.discoverServices());
  } else if (newState == BluetoothProfile
```

```
    .STATE_DISCONNECTED) {
    Log.i("NMG", "Gerät verloren");
  }
}
```

Aufgrund der immer größer werdenden Verbreitung von Bluetooth-LE-Hardware besteht ein valides Risiko, dass unser Scanner nicht kompatible Hardware ergreift und dort mitunter Schaden anrichtet. Wir umgehen dieses – zugegebenermaßen geringe, aber didaktisch günstige – Risiko durch den Vergleich der Services. Die Read- und Write-Buttons in der GUI werden nur dann aktiviert, wenn wir einen passenden Dienst finden:

```
@Override
public void onServicesDiscovered(BluetoothGatt gatt, int
status) {
  super.onServicesDiscovered(gatt, status);
  List<BluetoothGattService> myServices=
  gatt.getServices();
  for (BluetoothGattService gattService : myServices) {
    String uuid = gattService.getUuid().toString();
    if(uuid.compareTo
    ("abcbe138-a00c-6b8e-7d44-4b63a80170c3")==0)
    {//NMG-Dienst gefunden, GUI aktivieren
      myAct.runOnUiThread(new Runnable() {
      @Override
      public void run() {
      ...
    }
  });
...
```

Im Fall des Anklickens des Read-Buttons müssen wir mit der Beschaffung der Charaktistik beginnen. Es handelt sich dabei um einen asynchronen Prozess. Der Event Handler beschafft einen Verweis auf das Charakteristik-Objekt, um damit einen Lesebefehl abzusetzen:

```
@Override
public void onClick(View v)
{
  if(v==myReadBtn) {
    BluetoothGattService myGS=
    myGattConnection.getService(UUID.fromString
    ("abcbe138-a00c-6b8e-7d44-4b63a80170c3"));
    BluetoothGattCharacteristic myChara=
    myGS.getCharacteristic(UUID.fromString
    ("0acea172-2a76-69a7-4e49-302ed371f6a8"));
    myGattConnection.setCharacteristicNotification
    (myChara,true);
    if(myGattConnection.readCharacteristic(myChara)==
    true) {
      TextView myView = (TextView) findViewById
      (R.id.textView);
      myView.setText("Leseprozess beginnt");
    }
```

Android-x86 stellt keine großen Ansprüche an die Hardware (Bild 7)

```
  Else
  {
    TextView myView = (TextView) findViewById
    (R.id.textView);
    myView.setText("Start des Leseprozesses scheitert");
  }
}
```

Nach dem Abarbeiten des Leseprozesses folgt ein Aufruf von *onCharacteristicRead*. Die Methode liest die vom Server angelieferten Daten ein, um danach die Benutzerschnittstelle zu aktualisieren:

```
@Override
public void onCharacteristicRead(BluetoothGatt gatt,
final BluetoothGattCharacteristic characteristic,
int status)
{
  super.onCharacteristicRead(gatt, characteristic,
  status);
  myAct.runOnUiThread(new Runnable() {
    @Override
    public void run()
    {
      TextView myView=(TextView)myAct.findViewById
      (R.id.textView);
      myView.setText(characteristic.getStringValue(0));
    }
  });
}
```

Damit ist die App einsatzbereit. Schicken Sie den Client in die VM oder auf ein Telefon mit Android 4.4 und führen Sie

einen Scan durch. Nach dem Anklicken des Zielgeräts folgt ein Tap auf den Lesen-Button. Bild 5 zeigt das Resultat.

Aktivieren Sie den Funk

User schalten das Bluetoothmodul ihres Telefons gern aus, um Energie zu sparen. Dass die dabei zu erzielenden Ersparnisse normalerweise nicht signifikant sind, ist ein Thema für sich. Für uns als Entwickler ist die Situation nur insofern relevant, als der User bei Bedarf zum Einschalten des Transmitters aufgefordert werden muss.

Android sieht hierfür einen Standarddialog vor, der sich durch folgendes Codesnippet anwerfen lässt:

```
@Override
protected void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  setContentView(R.layout.activity_main);
  myManager = (BluetoothManager)
  getSystemService(Context.BLUETOOTH_SERVICE);
  myAdapter = myManager.getAdapter();
  ...
  if(!myAdapter.isEnabled())
  {
    Intent enableBtIntent = new Intent
    (BluetoothAdapter.ACTION_REQUEST_ENABLE);
    startActivityForResult(enableBtIntent, 232);
  }
  ...
```

Zwecks Vermeidung undefinierter Betriebszustände aktiviert unser Codebeispiel den Scanprozess nur dann, wenn der Transmitter des Telefons aktiv ist:

```
protected void onCreate(Bundle savedInstanceState) {
  ...
  if(myAdapter.isEnabled())
  {
    myScanCallback=new NMGScanCallback(this);
    scanDevices();
  }
}
```

Die als Host auftretende Activity kann über Erfolg beziehungsweise Misserfolg der Aktivierung informiert werden. Im Fall unseres Beispiels ist dazu folgender Code notwendig, der den Suchprozess bei erfolgreicher Aktivierung abermals anwirft:

```
@Override
protected void onActivityResult(int requestCode,
int resultCode, Intent data) {
  if(requestCode == 232) {
    if (resultCode == RESULT_OK) {
      myScanCallback = new NMGScanCallback(this);
      scanDevices();
    } else {
      Toast.makeText(this, "Aktivierung verweigert.
      Terminiere!", Toast.LENGTH_SHORT).show();
      finish();
    }
  }
}
```

Führen Sie *NMGGattClient* anschließend bei ausgeschaltetem Bluetooth-Transmitter aus, um sich am in Bild 6 gezeigten Prompt zu erfreuen.

Hilfreiches für Entwickler

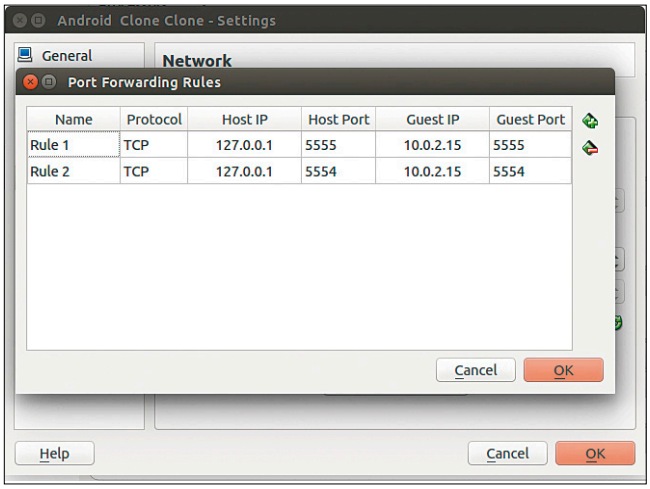
Der von Google angebotene Emulator unterstützt Bluetooth LE nicht. Die Einschränkungen lassen das Finden kompatibler Hardware in einen veritablen Spießrutenlauf ausarten. Wenn Sie ein als Host agierendes Telefon besitzen und auf der Suche nach einem Client sind, so können Sie auf eine VM zurückgreifen.

Als Funkmodul kommt dabei ein Bluetooth-4.0-USB-Stick zum Einsatz. Wer in Bezug auf Kompatibilität auf Nummer sicher gehen möchte, bestellt den in Entwicklerkreisen allgemein sehr populären BTA8000 von Cirago. Da die Anbindung durch den BlueZ BLE Stack für Linux erfolgt, funktionieren auch andere damit kompatible Dongles in der Regel problemlos. Der Autor nutzte einen handelsüblichen Dongle vom Typ Connect IT BT403 auf Basis des Harmony-Chipsatzes.

VirtualBox und USB-Supportpaket

Laden Sie im nächsten Schritt die aktuellste Version von VirtualBox herunter. Das USB-Supportpaket ist für das Freigeben des Dongles erforderlich und muss ebenfalls auf die Maschine wandern. Der Autor hat für diesen Artikel ein spezielles Image kompiliert, in dem zwecks Erhöhung der Kompatibilität mit diversen Dongles einige Sicherheitschecks im BTLE-Stack deaktiviert sind – laden Sie die rund 1,7 GByte große Datei unter www.tamoggemon.com/test/android_x86_btnmg2015.iso herunter.

Erzeugen Sie eine neue virtuelle Maschine mit den in Bild 7 gezeigten Einstellungen. Nach dem Start der VM wählen ▶



Ports: Die von adb benötigten Ports müssen mit dem Host verbunden werden (Bild 8)

Sie die Option *Live CD*. Die Installation erfolgt wie bei anderen unixoiden Betriebssystemen. Achten Sie darauf, die *.iso*-Datei nach der Installation aus der VM zu entfernen.

Während des rund zwei Minuten dauernden Bootprozesses können Sie die Eigenschaften der VM öffnen. In der Rubrik *USB* legen Sie durch Anklicken des Plus-Symbols einen neuen USB Device Filter an, der den USB-Dongle in Richtung der virtuellen Maschine weiterleitet. Ebendort müssen Sie die Netzwerkkarte konfigurieren. Klicken Sie auf *Port Forwarding* und richten Sie die in Bild 8 gezeigten Verdrahtungen ein.

Da das Betriebssystem keine VirtualBox-Erweiterungen mitbringt, müssen Sie die Mausintegration im Menü unter *Machine* und *Disable Mouse* Integration deaktivieren. Ab diesem Zeitpunkt ergreift die VM nach dem Anklicken Maus und Tastatur, die sich über die Host-Taste (normalerweise [Right Ctrl]) zurückerobern lassen.

Arbeiten Sie im nächsten Schritt den Konfigurationsassistenten ab, um das Tablet zu aktivieren. Bei erfolgreicher Installation lässt sich Bluetooth in den Einstellungen aktivieren; ein echtes Telefon kann die VM dann wie ein normales Smartphone finden. Aktivieren Sie zudem unter *Developer Options* die Möglichkeit zum Debugging per USB.

In VirtualBox lebende Android-Instanzen verhalten sich wie normale Unix-Betriebssysteme. Drücken Sie die Tastenkombination [Alt F1], um in die Konsole zu wechseln – die Eingabe von *adb tcpip 5555* sorgt dafür, dass der in der VM lebende adb Verbindungen per TCP/IP entgegennimmt.

Auf Seiten des Hosts ist danach die Eingabe folgendes Befehls notwendig, um die virtuelle Maschine als per *adb* ansprechbares Gerät einzubinden.

```
tamhan@TAMHAN14:/sdk/platform-tools$
./adb connect localhost:5555
* daemon not running. starting it now on port 5037 *
* daemon started successfully *
connected to localhost:5555
```

Im Moment sind unsere Beispiele nicht sonderlich interaktiv: Das Auslesen von am Server veröffentlichten Charakteristika mag für Sensoren ausreichen, geht für komplexere Aufgaben aber nicht weit genug.

Interaktivität

Wesentlich interessanter wird die Situation durch das Herausschreiben von Werten, die von *NMGGattServer* entgegengenommen werden können. Dazu muss die in *NMGGattServer* erfolgende Deklaration der Charakteristik Schreibrechte beinhalten. Adaptieren Sie *MainActivity.java*, um die zusätzlichen Flags per *OR* in die Parameter des Konstruktors zu schreiben:

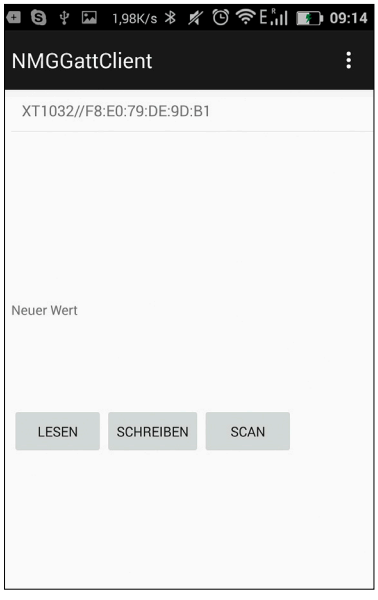
```
BluetoothGattService s = new
BluetoothGattService
(NMGConstants.ServiceUUID,
BluetoothGattService.SERVICE_TYPE_PRIMARY);
myCharacteristic = new BluetoothGattCharacteristic
(NMGConstants.CharacteristicUUID,
BluetoothGattCharacteristic.PROPERTY_READ |
BluetoothGattCharacteristic.PROPERTY_WRITE,
BluetoothGattCharacteristic.PERMISSION_READ|
BluetoothGattCharacteristic.PERMISSION_WRITE );
myCharacteristic.setValue("NMG Test");
s.addCharacteristic(myCharacteristic);
myServer.addService(s);
```

GATT kennt mehrere Schreiboperationen. Neben einem normalen Schreibvorgang mit Rückmeldung gibt es auch Befehlsvariationen, die eine Signatur oder auch eine verschlüsselte Verbindung voraussetzen. Für besonders energieverbrauchskritische Situationen bietet das Protokoll eine Variante an, bei der geschriebene Daten vom Empfänger nicht quittiert werden.

Wir beschränken uns hier aus Bequemlichkeitsgründen auf normale Writes. In *NMGGattServerObserver* müssen wir vom Client eingehende Werte entgegennehmen. Wie beim Lesen ist auch hier Handarbeit vom Entwickler notwendig:

```
@Override
public void onCharacteristicWriteRequest
(BluetoothDevice device, int requestId,
BluetoothGattCharacteristic characteristic, boolean
preparedWrite, boolean responseNeeded, int offset,
byte[] value) {
  updateView("Schreibrequest ist eingegangen");
  super.onCharacteristicWriteRequest(device,
  requestId, characteristic, preparedWrite,
  responseNeeded, offset, value);
  characteristic.setValue(value);
  myActivity.myServer.sendResponse(device, requestId,
```

Der Server nimmt den Schreibbefehl entgegen (Bild 9)



Textbox: Die neuen Informationen scheinen in der Textbox des Clients auf (Bild 10)

```
  BluetoothGatt.GATT_SUCCESS, offset,
  characteristic.getValue());
}
```

Unsere Methode beginnt damit, den Schreib-Request in der Textbox anzukündigen. Im nächsten Schritt folgt ein Aufruf der für das Setzen der Charakteristik notwendigen Funktion. Die Daten wandern automatisch ins Betriebssystem, wo sie von Clients abgeerntet werden können.

Die Anpassung von *NMGGattClient* beginnt im Event Handler der Knöpfe: Das Drücken des *Schreiben*-Knopfs führt zum Absetzen eines an den Server gerichteten Schreibbefehls:

```
public void onClick(View v) {
  ...
  else if (v==myWriteBtn)
  {
    BluetoothGattService myGS=
    myGattConnection.getService(UUID.fromString
    ("abcbe138-a00c-6b8e-7d44-4b63a80170c3"));
    BluetoothGattCharacteristic myChara=
    myGS.getCharacteristic(UUID.fromString
    ("0acea172-2a76-69a7-4e49-302ed371f6a8"));
    myChara.setValue("Neuer Wert");
    myChara.setWriteType
    (BluetoothGattCharacteristic.WRITE_TYPE_DEFAULT);
    if(myGattConnection.writeCharacteristic(myChara)==
    true) {
      TextView myView = (TextView) findViewById
      (R.id.textView);
      myView.setText("Schreibprozess beginnt");
    }
    else
    {
      TextView myView = (TextView) findViewById
      (R.id.textView);
      myView.setText("Start des Schreibprozesses
      scheitert");
    }
  }
```

Damit ist die Arbeit am Schreib-Feature abgeschlossen. Starten Sie die beiden Apps auf kompatibler Hardware und klicken Sie auf den *Schreiben*- und danach auf den *Lesen*-Button des Clients. Bild 9 und Bild 10 zeigen, wie die beiden Programme erfolgreich interagieren.

Entfernungsmessung mit Bluetooth LE

Der Austausch von Daten per KV-Speicher ist reizvoll, deckt aber nur einen Teil der Möglichkeiten von Bluetooth LE ab. Androids BTLE-Stack liefert im Rahmen der Suche Informationen über die Signalstärke, die sich zur Indoor-Navigation einsetzen lassen.

Bei der Arbeit mit Beacons gibt es zwei verbreitete Konzepte. Methode 1 nutzt das aus der Ortung von GSM-Telefonen bekannte Verfahren der Dreieckspeilung – der Aufenthaltsort des Telefons lässt sich durch die drei von den Sendern gezogenen Kreise bestimmen. Als Alternative dazu bietet sich die Nutzung von Beacons mit sehr schwacher Sendeleistung an. Wenn ein Telefon die vom Sender ausgehenden Signale empfangen kann, so dürfte es in nächster Nähe sein.

In beiden Fällen gilt: Ein iBeacon ist ein Gerät, das permanent Advertising Packets aussendet. Die Payload beginnt dabei stets mit der Bitsequenz 02 01 06 1A FF 4C 00 02 15: statt 06 sind im dritten Byte auch andere Flag-Kombinationen zulässig. Darauf folgen 16 Bytes, die Apple als Proximity ID bezeichnet. Sie beschreibt den Betreiber des Beacons. Alle Tesco-Supermärkte würden sich bei normgerechter Implementierung ein- und dieselbe ID teilen. Die eigentliche Unterscheidung der Beacons erfolgt über vier Bytes, die in Major und Minor unterteilt werden: Major würde dabei den individuellen Supermarkt beschreiben, während Minor den Standort des individuellen Beacons beschreibt.

Da Google die vom Hauptkonkurrenten Apple entwickelte Technologie in seinen Telefonen natürlich nicht unterstützt, hat sich die von Radius Networks entwickelte Android Beacon Library als Quasistandard etabliert (https://altbeacon.github.io/android-beacon-library).

Dank Gradle erfolgt die Einbindung in eigene Projekte mit minimalem Aufwand. Öffnen Sie die für das Modul *App* zuständige Version der *build.gradle*-Datei und adaptieren Sie das Feld *dependencies:*

```
dependencies {
  compile fileTree(dir: 'libs', include: ['*.jar'])
  testCompile 'junit:junit:4.12'
  compile 'com.android.support:appcompat-v7:23.0.1'
  compile 'com.android.support:design:23.0.1'
  compile 'org.altbeacon:android-beacon-library:2+'
}
```

Im Rahmen der nächsten Kompilation lädt Gradle die fehlenden Ressourcen automatisch aus dem Internet herunter. Als Nächstes können Sie die unter https://altbeacon.github.io/android-beacon-library/samples.html bereitgestellten Anweisungen befolgen, um Ihre Applikation zur Kommunikation mit Beacons zu befähigen.

An dieser Stelle sei noch eine Anmerkung zu einer Besonderheit der Bibliothek erlaubt. Radius begegnet der Vielzahl der am Markt befindlichen Beaconformate mit einem kleinen Kniff – die *BeaconParser*-Klasse nimmt einen Formatstring entgegen, der die zu generierenden Pakete beschreibt. ■

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

DAS BATTERY STATUS API

Blick in den API-Dschungel

In dieser Artikelreihe geht es dieses Mal um das Battery Status API.

Der Siegeszug von mobilen Endgeräten hat für Webentwickler zur Folge, dass man sich bei der Entwicklung einer Webanwendung noch mehr Gedanken machen muss als noch vor einigen Jahren.

Welche Auflösung steht zur Verfügung? Wie muss die Oberfläche der Anwendung diesbezüglich angepasst werden? Ist der Nutzer online oder offline? Müssen eventuell Daten gecacht oder mit dem Server synchronisiert werden? Sollen standortbezogene Informationen in der Anwendung dargestellt werden, oder liefert das verwendete Endgerät keine standortbezogenen Daten?

Auf der einen Seite ist dadurch die Implementierung von Webanwendungen komplexer geworden, auf der anderen Seite bieten sich dem Entwickler bezüglich Usability und User Experience eine ganze Reihe neuer Möglichkeiten.

Ein Web API, die diesbezüglich interessant ist, ist das sogenannte Battery Status API (www.w3.org/TR/battery-status). Über dieses API hat man nämlich die Möglichkeit, auf Batterie- beziehungsweise Akkuinformationen des jeweiligen Endgeräts zuzugreifen und somit etwa abhängig vom Ladestatus des Akkus die Anwendung entsprechend anzupassen.

Status und Browsersupport

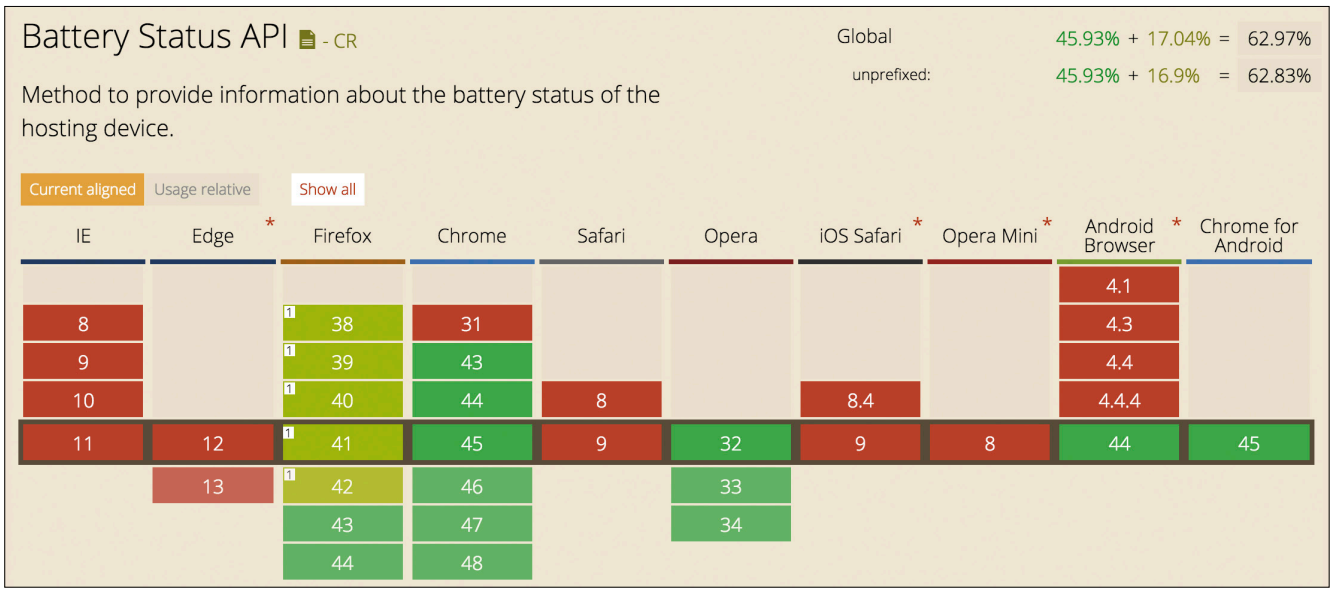
Das API befindet sich momentan im Status Candidate Recommendation und wird in den aktuellen Versionen von Firefox, Chrome und Opera unterstützt. Der Internet Explorer, dessen Nachfolger Edge sowie Safari bieten momentan noch keine Unterstützung dafür an (Bild 1).

Das API definiert ein neues Interface, das Interface *BatteryManager*, über das der Zugriff auf verschiedene Akkuinformationen möglich ist. Zugriff auf eine Objektinstanz vom Typ *BatteryManager* erhält man über die (ebenfalls neue) Methode *getBattery()* des *navigator*-Objekts.

Um zu überprüfen, ob der jeweilige Browser das Battery Status API unterstützt oder nicht, reicht es daher wie in Listing 1 gezeigt aus, zu prüfen, ob das *navigator*-Objekt über die

Listing 1: Browsersupport für das Battery Status API prüfen

```
if (navigator.getBattery) {
  // Battery Status API unterstützt
} else {
  // Battery Status API nicht unterstützt
}
// Alternativ dazu kann folgender Code genutzt werden:
if ('getBattery' in navigator) {
  // Battery Status API unterstützt
} else {
  // Battery Status API nicht unterstützt
}
```



Browserunterstützung: Derzeit wird das Battery Status API noch nicht von allen größeren Browsern unterstützt (Bild 1)

Methode *getBattery()* verfügt (übrigens sahen ältere Versionen des API vor, den *BatteryManager* über die Eigenschaft *battery* am *navigator*-Objekt zur Verfügung zu stellen. Auch wenn diese Information in vielen Online-Tutorials zu finden ist, handelt es sich hierbei um eine veraltete Version des API).

Verwendung des API

Das *BatteryManager*-Interface stellt vier Eigenschaften bereit. Die Eigenschaft *charging* (vom Typ *boolean)* gibt Auskunft darüber, ob die Batterie momentan aufgeladen wird oder nicht. Über die Eigenschaft *chargingTime* erhält man eine Zeitangabe (Angabe erfolgt in Sekunden), wie lange es dauert, bis die Batterie komplett aufgeladen ist. Für den Fall, dass die Batterie bereits komplett aufgeladen ist, hat diese Eigenschaft den Wert *0*.

Wird die Batterie momentan nicht aufgeladen, hat die Eigenschaft dagegen den Wert *Infinity*. Analog zu *chargingTime* gibt die Eigenschaft *dischargingTime* die Zeit an (ebenfalls in Sekunden), die es dauert, bis die Batterie vollständig entladen ist (wird die Batterie momentan geladen, hat *dischargingTime* den Wert *Infinity)*. Zu guter Letzt gibt die Eigenschaft *level* den Batteriestand als Fließkommazahl zwischen *0* (nicht geladen) und *1* (vollständig geladen) an. ▶

Tabelle 1: Die verschiedenen Eigenschaften von BatteryManager

Eigenschaft	Beschreibung
charging	Gibt an, ob der Akku geladen wird oder nicht
chargingTime	Gibt die Zeit an, die nötig ist, bis der Akku vollständig geladen ist
dischargingTime	Gibt die Zeit an, die nötig ist, bis der Akku vollständig entladen ist
level	Gibt den Akkuladestand als Fließkommazahl zwischen 0 (nicht geladen) und 1 (vollständig geladen) an

Tabelle 2: Die verschiedenen Ereignisse rund um den Akkuladestand

Event	Beschreibung
chargingchange	Wird ausgelöst, wenn das Endgerät seinen Ladestatus ändert
levelchange	Wird ausgelöst, wenn sich der Ladestand ändert
charging-timechange	Wird ausgelöst, wenn sich die Zeit ändert, die nötig ist, bis der Akku den vollen Ladestatus erreicht hat
dischargingtime-change	Wird ausgelöst, wenn sich die Zeit ändert, die nötig ist, bis der Akku leer ist

Listing 2: Verwenden des Battery Status API

```
<!DOCTYPE html>
<html>
<head lang="en">
  <meta charset="UTF-8">
  <title>Beispiel</title>
  <link rel="stylesheet" href="styles/main.css"
  type="text/css">
</head>

<body id="battery-status" class="battery-charge">
<table>
  <tr>
    <th>Lädt:</th>
    <td id="charge"></td>
  </tr>
  <tr>
    <th>Ladezeit:</th>
    <td id="charging-time"></td>
  </tr>
  <tr>
    <th>Verbleibende Zeit:</th>
    <td id="discharging-time"></td>
  </tr>
  <tr>
    <th>Batteriestatus:</th>
    <td id="battery-level"></td>
  </tr>
</table>
<script src="scripts/main.js"></script>
</body>
</html>

// Inhalt der Datei main.js
if (navigator.getBattery) {
  function displayBatteryStatus(battery) {
    document.getElementById('charge').innerHTML =
    (battery.charging ? 'Ja' : 'Nein');
    document.getElementById('charging-time').innerHTML
    = battery.chargingTime;
    document.getElementById
    ('discharging-time').innerHTML =
    battery.dischargingTime;
    document.getElementById
    ('battery-level').innerHTML = battery.level;
  }
  navigator.getBattery().then(function (battery)
  {
    displayBatteryStatus(battery);
  });
}
```

Theme: Wird der Akku geladen oder ist er bereits vollständig geladen, so wird das normale Theme verwendet (Bild 2)

Dunkel: Wird der Akku nicht geladen, wird das normale Theme durch ein dunkleres Theme ersetzt (Bild 3)

Ein einfaches Beispiel für die Verwendung des API finden Sie in Listing 2. Wie Sie sehen, liefert die Methode *getBattery()* nicht direkt ein *BatteryManager*-Objekt zurück, sondern ein *Promise*-Objekt. Ruft man auf diesem wiederum die Methode *then()* auf, gelangt man in der übergebenen Callback-Funktion schließlich an das gewünschte *BatteryManager*-Objekt (Tabelle 1).

Neben den genannten Eigenschaften können dem *BatteryManager* über die Methode *addEventListener()* Event-Listener für vier verschiedene Events zugewiesen werden: Das Event *chargingchange* wird ausgelöst, wenn sich der Ladestatus ändert (das heißt, von *ladend* zu *nicht ladend* oder von *nicht ladend* zu *ladend)*, das Event *levelchange*, wenn sich der Ladestand ändert, und die Events *chargingtimechange* und *dischargingtimechange*, wenn sich die Zeit ändert, bis zu der der Akku vollständig geladen *(chargingtimechange)* beziehungsweise entladen *(dischargingtimechange)* ist. Tabelle 2 gibt einen Überblick über die verschiedenen Ereignisse rund um den Akkuladestand. Ein Beispiel hierzu sehen Sie in Listing 3.

CSS abhängig vom Batteriestand ändern

Je nachdem, mit welchem Endgerät man eine Webseite besucht, kann es sein, dass die auf der Webseite verwendeten Farben beziehungsweise deren Helligkeit Auswirkungen auf die Akkulaufzeit des Endgeräts haben.

Dies ist zum Beispiel bei Endgeräten wie der Samsung-Galaxy-Reihe der Fall, welche die sogenannte OLED-Technologie für ihr Display verwenden, da hier jedes einzelne Pixel selbst leuchtet.

Listing 3: Über Event Listener kann auf verschiedene Ereignisse reagiert werden

```
if (navigator.getBattery) {
  function displayBatteryStatus(battery)   {
    document.getElementById('charge').innerHTML =
    (battery.charging ? 'Ja' : 'Nein');
    document.getElementById('charging-time').innerHTML
    = battery.chargingTime;
    document.getElementById
    ('discharging-time').innerHTML =
    battery.dischargingTime;
    document.getElementById
    ('battery-level').innerHTML = battery.level;
  }

  function chargingChangeHandler(event) {
    console.log('chargingchange');
    displayBatteryStatus(event.target);
  }

  function chargingTimeChangeHandler(event) {
    console.log('chargingtimechange');
    displayBatteryStatus(event.target);
  }

  function dischargingTimeChangeHandler(event) {
    console.log('dischargingtimechange');
    displayBatteryStatus(event.target);
  }

  function levelChangeHandler(event) {
    console.log('levelchange');
    displayBatteryStatus(event.target);
  }

  navigator.getBattery().then(function (battery) {
    displayBatteryStatus(battery);
    battery.addEventListener('chargingchange',
    chargingChangeHandler);
    battery.addEventListener('chargingtimechange',
    chargingTimeChangeHandler);
    battery.addEventListener('dischargingtimechange',
    dischargingTimeChangeHandler);
    battery.addEventListener('levelchange',
    levelChangeHandler);
  });
}
```

Listing 4: Das Bootstrap-Theme wird abhängig vom Akkuladestand angepasst

```
<!DOCTYPE html>
<html>
<head lang="en">
  <meta charset="UTF-8">
  <title>Beispiel</title>
  <link rel="stylesheet" href="https://
  maxcdn.bootstrapcdn.com/bootstrap/3.3.5/css/
  bootstrap.min.css">
  <link id="theme" rel="stylesheet"
  href="styles/default-theme.min.css">
</head>

<body id="battery-status" class="battery-charge">
  <div class="container">
    <ul class="nav nav-pills nav-stacked">
      <li><a href="#">Home</a></li>
      <li class="active"><a href="#">Projects</a>
      </li>
      <li><a href="#">Services</a></li>
      <li><a href="#">About Us</a></li>
      <li><a href="#">Contact</a></li>
    </ul>
  </div>
<script src="scripts/main.js"></script>
</body>
</html>

// Inhalt der Datei main.js
function init() {
  if (navigator.getBattery) {
    var normalTheme = 'default-theme.min.css';
    var darkTheme = 'dark-theme.min.css';

    function applyCSS(battery)
    {
      if (battery.charging) {
        document.getElementById('theme').href =
        'styles/' + normalTheme;
      }
      else {
        document.getElementById('theme').href =
        'styles/' + darkTheme;
      }
    }

    function chargingChangeHandler(event)
    {
      applyCSS(event.target);
    }
    navigator.getBattery().then(function (battery)
    {
      applyCSS(battery);
      battery.addEventListener('chargingchange',
      chargingChangeHandler);
      battery.addEventListener('levelchange',
      chargingChangeHandler);
      battery.addEventListener('chargingtimechange',
      chargingChangeHandler);
      battery.addEventListener
      ('dischargingtimechange',
      chargingChangeHandler);
    });
  }
}
document.addEventListener('DOMContentLoaded', init);
```

Bei Endgeräten, die die LCD-Technologie verwenden, wie beispielsweise die verschiedenen iPhones, bei der unabhängig von der Farbe einzelner Pixel immer der gesamte Hintergrund leuchtet, hat die Farbe der Pixel dagegen keinen Einfluss auf die Akkulaufzeit.

Ein Anwendungsbeispiel für die Verwendung des Battery Status API könnte es also sein, das Theme einer Webseite abhängig vom Akkuladestands anzupassen (Bild 2): Ist der Akku noch ausreichend geladen, wird das normale Theme verwendet. Liegt der Akkuladestand unterhalb eines bestimmten Schwellenwerts, wird auf ein energiesparenderes Theme gewechselt (Bild 3).

Listing 4 zeigt dazu ein Beispiel, wobei hier nicht abhängig vom Akkuladestand zwischen den Themes gewechselt wird, sondern abhängig davon, ob der Akku gerade geladen wird oder nicht. Das Beispiel lässt sich sogar auf einem Laptop ausprobieren: Trennen Sie diesen einfach vom Strom, und schon sollte das Theme gewechselt werden (es kann allerdings durchaus etwas länger dauern, bis das *chargingchange*-Event ausgelöst wird).

Fazit

Das Battery Status API ermöglicht es Programmierern, auf Akkuinformationen eines Endgeräts zuzugreifen. Das API ist nicht komplex, trägt aber dazu bei, dass Webanwendungen noch nutzerfreundlicher programmiert werden können. Ein konkretes Beispiel wird in diesem Artikel gezeigt: Das CSS des Layouts wird abhängig vom aktuellen Akkuladestand dynamisch angepasst. Momentan unterstützen die Browser Firefox, Chrome und Opera das Battery Status API. ■

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.

IOS: CORE GRAPHICS API

Grafiken erzeugen

Das Core Graphics API ist eine Schnittstelle zum Zeichnen geometrischer Elemente in einer View.

Foto: Shutterstock / Siarhei Tolak

Neben Controls sollen in der eigenen App noch zusätzliche grafische Elemente angezeigt werden? Das geht unter iOS oder OS X mit dem Core Graphics API. Hierbei handelt es sich um eine Bibliothek von Funktionen, die auf der Quartz-Engine des jeweiligen Betriebssystems aufsetzt.

Die Quartz-Engine ermöglicht das Rendering von 2D-Elementen. Um mit Quartz arbeiten zu können, wird das entsprechende API (Core Graphics) verwendet. Das Zeichnen beginnt mit der Erstellung einer sogenannten Zeichenfläche, dem Graphics Context. Auf dieser Zeichenfläche werden im späteren Verlauf die gewünschten Elemente erzeugt. Neben der Erzeugung von Grafiken ist es beispielsweise auch möglich, auf Basis von Core Graphics ein PDF-Dokument zu erzeugen.

Vorbereitung

Vor dem Zeichnen von geometrischen Figuren muss erst einmal ein Graphics Context erzeugt werden. Hierbei handelt es sich um eine Fläche, auf der später die Figuren angezeigt werden. Dieses Objekt wird in einer App mittels einer zusätzlichen Klasse erstellt.

Listing 1: Klasse vorbereiten

```
import UIKit

class uiView: UIView {

  override init(frame: CGRect) {
    super.init(frame: frame)
  }

  required init?(coder
    aDecoder: NSCoder) {
      fatalError("init(coder:) wurde nicht
      implementiert!")
  }

  override func drawRect(rect:
    CGRect) {
  }

//Quellcode entfernt ...
}
```

In einem Projekt vom Typ *Single View Application* wird hierzu eine neue Klasse, die von der Klasse *UIView* abgeleitet wird, angelegt. Im Projekt kann eine entsprechende Klasse via *File, New File, iOS* und *Source* und dort mittels Auswahl der Vorlage *Cocoa Touch Class* generiert werden.

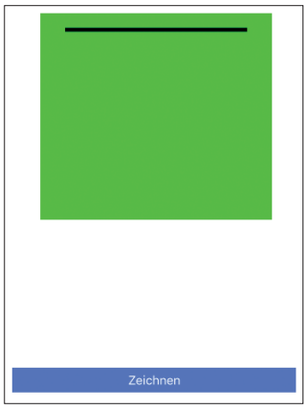
Bei der Anlage der neuen Klasse hilft der entsprechende Dialog von Xcode. Hier muss man zuerst einen passenden Namen wählen und als Subclass-Typ *UIView* wählen. Im Beispiel wird Swift als Sprache verwendet. Nach Anlage der

Listing 2: Eine Linie zeichnen

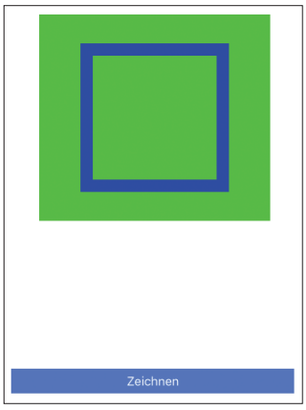
```
func drawLine() {

  let ctx =
    UIGraphicsGetCurrentContext()

  CGContextBeginPath(ctx)

  CGContextMoveToPoint(ctx, 30.0
  , 20.0)
  CGContextAddLineToPoint(ctx, 250.0, 20.0)

  CGContextSetLineWidth(ctx, 5)
  CGContextClosePath(ctx)
  CGContextStrokePath(ctx)
}
```

neuen Klasse muss diese um einige Methoden erweitert werden, welche für die Vorbereitung der Zeichenfläche notwendig sind. Überschrieben werden unter anderem die Methoden *init* und *drawRect* (Listing 1).

Im ersten Schritt wird der Initialisierer der Klasse, die Methode *init*, aufgerufen. Diese Methode erwartet ein Objekt vom Typ *CGRect*. Dieses Objekt wird später als Zeichenfläche verwendet. Im Initialisierer wird das Objekt vom Typ *CGRect* übernommen und im weiteren Verlauf der überschriebenen Methode *drawRect* als Parameter übergeben.



Linie: Eine einfache Linie wird gezeichnet (Bild 1)



Rechteck: Ein Rechteck wird gezeichnet (Bild 2)

Erst innerhalb von *drawRect* werden dann im nächsten Schritt verschiedene Methoden zum Zeichnen aufgerufen. Es gibt noch einen weiteren Initialisierer im Code, der mit dem Schlüsselwort *required* gekennzeichnet ist.

Durch die Verwendung wird sichergestellt, dass auch dieser Initialisierer aufgerufen wird. Im Fall eines Fehlers wird dann eine entsprechende Meldung ausgegeben.

Graphics Context und Line

Nach diesen Vorbereitungen ist man nicht mehr weit davon entfernt, mit dem Zeichnen zu beginnen. Innerhalb der Methode *drawLine* soll eine einfache gerade Linie gezeichnet werden (Listing 2). Für das Zeichnen wird, wie bereits erwähnt, ein Graphics-Context-Objekt benötigt. Dessen Anlage erfolgt durch einen Aufruf der Methode *UIGraphicsGetCurrentContext*.

Der Zeichenvorgang beginnt durch Aufruf der Methode *CGContextBeginPath*, dieser Methode wird als Parameter das Graphics-Context-Objekt *ctx* übergeben. Eine Zeichenfläche für die Ausgabe der Grafik wurde damit festgelegt.

Anschließend muss bestimmt werden, wo in der View mit dem Zeichnen begonnen werden soll. Hierzu wird die Methode *CGContextMovieToPoint* verwendet. Dieser Methode werden neben dem Graphics Context noch die entsprechenden x- und y-Koordinaten übergeben, mit denen der Startpunkt festgelegt wird. Das Zeichnen der Linie wird mit der Methode *CGContextAddLineToPoint* durchgeführt. Dieser Methode wird wieder das Graphics-Context-Objekt übergeben, und dazu die x- und y-Koordinaten, die den Endpunkt der Linie angeben.

Es fehlt noch die Festlegung, mit welcher (Strich-)Stärke die Linie gezeichnet werden soll. Dies wird durch Aufruf der Methode *CGContextSetLineWidth* festgelegt. Als Parameter werden hier das Graphics-Context-Objekt und ein Wert vom Typ *CGFloat* übergeben.

Zum Abschluss werden dann die Methoden *CGContextClosePath* und *CGContextStrokePath* aufgerufen. Beiden Methoden wird als Parameter das Graphics-Context-Objekt übergeben. Die Methode *CGContextClosePath* schließt den Ausgabepfad, und *CGContextStrokePath* sorgt dafür, dass die Linie dann gezeichnet wird (Bild 1). Das sind bereits die wesentlichen Schritte, um zu zeichnen.

Listing 3: Festlegung eines Mittelpunkts

```
func drawRectangle() {
  let center = CGPointMake(
    self.frame.size.width / 2.0,
    self.frame.size.height / 2.0)
  let rectangleWidth:CGFloat = 150.0
  let rectangleHeight:CGFloat = 150.0
  let ctx =
    UIGraphicsGetCurrentContext()

  CGContextAddRect(ctx,
    CGRectMake(center.x - (0.5 *
      rectangleWidth), center.y - (0.5
        * rectangleHeight),
          rectangleWidth,
            rectangleHeight))
  CGContextSetLineWidth(ctx, 30)
  CGContextSetStrokeColorWithColor(
    ctx, UIColor.blueColor().CGColor)
  CGContextStrokePath(ctx)

}
```

Rechteck

Die nächste Figur, die gezeichnet werden soll, ist ein Rechteck. Hierfür gibt es eine eigene Methode, sodass es nicht erforderlich ist, vier einzelne Linien zu zeichnen. In Listing 3 beginnt die Erstellung der Zeichnung mit der Festlegung eines Mittelpunkts. Die ersten Zeilen werden dazu verwendet, die Position des zu zeichnenden Rechtecks sowie dessen Dimensionen zu bestimmten. Anschließend wird ein Graphics-Context-Objekt erzeugt. Das Zeichnen des Rechtecks beginnt mit dem Aufruf der Methode *CGContextAddRect*.

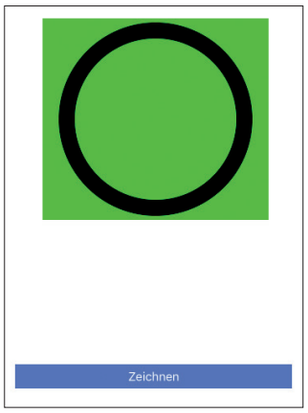
Unter Berücksichtigung der (späteren) Position werden die Höhe und die Breite des Rechtecks als Parameter der Methode *CGContextAddRect* übergeben. Die Stärke der Linie des Rechtecks wird, wie im letzten Beispiel, mit der Methode *CGContextSetLineWidth* festgelegt. Außerdem wird noch die Farbe festgelegt, in der der Rahmen des Rechtecks gezeichnet wird. Hierzu wird die Methode *CGContextSetStroke-* ▶

ColorWithColor aufgerufen und neben dem Graphics-Context-Objekt noch die zu verwendende Farbe übergeben. Abschließend wird zum Zeichnen des Objekts wieder die Methode *CGContextStrokePath* aufgerufen und das Rechteck wird nun angezeigt (Bild 2).

Auch das Zeichnen eines Kreises ist unter Verwendung der entsprechenden Anweisungen nicht weiter schwierig. Hierbei müssen natürlich zusätzliche Methoden und Parameter verwendet werden (Listing 4).

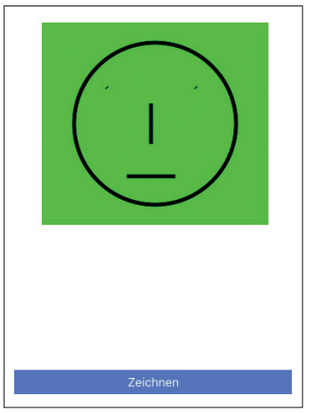
Um einen Kreis zu zeichnen, wird im ersten Schritt wieder der Mittelpunkt mit der Methode *CGPointMake* bestimmt. Als Nächstes wird ein Graphics-Context-Objekt erzeugt und es wird die Methode *CGContextBeginPath* aufgerufen. Die Stärke, in welcher der Kreis gezeichnet wird, wird wieder durch einen Aufruf der Methode *CGContextSetLineWidth* festgelegt.

Mittels der zuvor angelegten Variablen *center* werden nun der aktuelle x- und anschließend der y-Wert abgefragt. Außerdem wird der Radius festgelegt sowie der Endwinkel berechnet. Die ermittelten Informationen werden im Anschluss der Methode *CGContextAddArc* übergeben. Gezeichnet wird das Objekt abschließend durch einen Aufruf der Methode *CGContextStrokePath* (Bild 3).



Kreis: Der gezeichnete Kreis (Bild 3)

Bisher wurden nur sehr einfache geometrische Figuren gezeichnet. Es ist klar, dass die Erstellung einer etwas aufwendigeren Zeichnung auch ein wenig mehr Quellcode erfordert. Sicherlich kennen Sie den Reim »Punkt, Punkt, Komma, Strich, fertig ist das Mondgesicht«. Dieser Reim soll im letzten Beispiel einmal in Code gegossen werden. Vorab sollte die Überlegung stehen, welche Elemente zum Zeichnen des Objekts benötigt werden. Es ist klar, dass neben einem Kreis als weiteres Element Linien zur Darstellung verwendet werden (Listing 5).



Gesicht: Das fertige Mondgesicht (Bild 4)

Da für den Rand des Gesichts ein Kreis gezeichnet werden muss, wird im ersten Schritt der Mittelpunkt bestimmt. Hierzu wird abermals die Methode *CGPointMake* verwendet. Anschließend wird das Graphics-Context-Objekt erzeugt und die Methoden *CGContextBeginPath* sowie *CGContextSetLineWidth* werden mit den jeweiligen Parametern aufgerufen. Anschließend werden, wie im vorigen Beispiel, die benötigten Informationen ermittelt, um den Kreis zeichnen zu können, und mit einem Aufruf von *CGContextAddArc* wird dieser dann gezeichnet.

Um die Augen, Nase sowie den Mund zu zeichnen, werden dann jeweils im Wechsel die Methoden *CGContextMovieToPoint*, *CGContextAddLineToPoint* sowie *CGContextSetLineWidth* mit den benötigten Koordinaten als Parameter aufgerufen. Mit dem Aufruf der Methode *CGContextStrokePath* wird abschließend das Mondgesicht gezeichnet (Bild 4).

Listing 4: Zeichnen eines Kreises

```
func drawCircle() {
  let center = CGPointMake(
    self.frame.size.width / 2,
      self.frame.size.height / 2)
  let ctx =
    UIGraphicsGetCurrentContext()
  CGContextBeginPath(ctx)
  CGContextSetLineWidth(ctx, 20)
  let x:CGFloat = center.x
  let y:CGFloat = center.y
  let radius: CGFloat = 110.0
  let endAngle: CGFloat = CGFloat(2 *
    M_PI)
  CGContextAddArc(ctx, x, y, radius, 0,
    endAngle, 0)
  CGContextStrokePath(ctx)
}
```

Links zum Thema

- Framework Reference
  https://developer.apple.com

Fazit

Sind die entsprechenden Anweisungen zum Zeichnen sowie die Verwendung des Graphics-Context-Objekts bekannt, dann ist das Zeichnen von Figuren innerhalb von iOS- oder auch OS-X-Apps kein Problem.

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

SMART DATA
Developer Conference
Big Data Analytics

- 18.04.2016 – Konferenz
- 19.04.2016 – Workshops

Novotel München City

Die SMART DATA Developer Conference macht Softwareentwickler mit den Herausforderungen von Big Data vertraut.

- In den Konferenz-Sessions erlangen Sie Wissen zu Speicherung, Analyse, Plattformen und Tools.
- Die Workshops bieten intensives Training in den Technologien Multi-Model-Database mit CQRS und OrientDB, Microsoft Azure und Architektur sowie Smart Development.

X-Mas Special
bis 6. Januar 2016
bis zu
300,– EUR
sparen!

Das Advisory Board der SMART DATA Developer Conference:

Sascha Dittmann, Microsoft Deutschland GmbH

Robert Eichenseer, Microsoft Corp.

Oliver B. Fischer, E-Post Development GmbH

Constantin Klein, Freudenberg IT SE & Co. KG

Michael Nolting, Sevenval Technologies GmbH

Stefan Papp, Teradata Operations Inc.

Agnzieszka Walorska, Creative Construction Heroes GmbH

smart-data-developer.de **#smartddc** **SMARTDATADeveloperConference**

Präsentiert von: dotnetpro, web & mobile DEVELOPER

Veranstalter: developer media, Neue Mediengesellschaft Ulm mbH

Partner-Konferenz: SMART DATA Summit

SMART DATA
Developer Conference
Big Data Analytics

Konferenz-Programm: 18. April 2016

09.00 Uhr

Clouds in the Wind – Big und Smart Data in der Cloud und trotzdem beweglich • *Michael Nolting*

Smart und Big Data sind die Trendthemen heutzutage. Aber welche Technologie und welcher Anbieter sind für den speziellen Anwendungsfall am besten geeignet? Die entwickelte Entscheidungsmatrix soll den Zuhörer mündig machen, die im Markt verfügbaren Technologie-Stacks schnell zu vergleichen und die für den eigenen Anwendungsfall am besten geeigneten zu finden.

10.00 Uhr — Kaffeepause

10.30 Uhr

Smart Data mit .NET

Performance trotz Entity Framework
• *André Krämer*

Ist das Entity Framework wirklich langsam? JA! In dieser Session werden wir uns ansehen, was das Entity Framework langsam macht und was wir dagegen unternehmen können!

Best Practices

Datenqualität für Entwickler
• *Werner Keil*

Big Data ohne Datenqualität wird chaotisch und bedeutungslos. Die Session bietet einen Überblick bekannter Standards, sowie deren Unterstützung durch Sprachen, APIs etc.

11.30 Uhr

Smart Data mit .NET

Datenpipeline-Komponenten für die SSIS
• *Thomas Worm*

Die SQL Server Integration Services ermöglichen die Integration von Datenbanken in vorhandene IT-Landschaften. Die Session behandelt Framework und Eigenheiten der SSIS.

Best Practices

Daten entlang der Seidenstraße
• *Jan Fellien*

Die Session stellt die Multi-Model-Graphen-Datenbank OrientDB mit node.js vor. Sie kann in Documents wie in Graphen speichern, was die Anwendung im Alltag sehr vereinfacht.

12.30 Uhr — Mittagspause

13.30 Uhr

Smart Data mit .NET

Datenquellen mit F# Data und Deedle
• *Stefan Lieser*

Mit diesen beiden Open-Source-Bibliotheken ist der Zugriff auf Datenquellen sowie ihre Analyse und Aggregation auf einfache Weise möglich.

Best Practices

Das Analytics Framework BRAIN Reflex
• *Rupert Steffner und Jan Lendholt*

Fokus der Session ist das Streaming Analytics Framework und dessen containerized Micro-Service-Architektur. Gezeigt werden Design Patterns und Use Cases.

14.30 Uhr

Smart Data mit .NET

Interaktive Analysen mit Apache Spark
• *Olivia Klose und Sascha Dittmann*

Bei interaktiven Datenabfragen schwächeln die meisten Big Data Technologien. Diese Session zeigt die wichtigsten Grundlagen der In-Memory Technologie anhand von Codebeispielen.

Best Practices

Mehr als SQL Server in the Cloud
• *Constantin Klein*

Gewinnen Sie in dieser Session einen Überblick über die Microsoft Cloud Data Platform und die Möglichkeiten beider Bereiche „Data & Storage" und „Analytics".

15.30 Uhr — Kaffeepause

16.00 Uhr

Smart Data mit .NET

Looking for the meaning to our data
• *Diego Poza*

This talk is going to get deep into real world scenarios using Azure Machine Learning. Build from scratch a predictive model, comparing algrorithms and adding custom modules.

Mensch & Maschine

The UX of Data: Responsive Visualisierung
• *Peter Rozek*

Big Data ist nicht gleich Smart Data. Interaktive und skalierbare Datenvisualisierung ist der nächste Schritt, um Daten für den Nutzer nutzerfreundlich aufzubereiten.

17.00 Uhr bis 18.00 Uhr

Smart Data mit .NET

Hey Cortana - let's talk about Analytics!
• *Olivia Klose und Marcel Tilly*

Ein Tiefflug über den Cortana Analytics Stack: Wie spielen die einzelnen Dienste, wie EventHub, Stream Analytics, Azure Data Lake, Azure Machine Learning und PowerBI zusammen?

Mensch & Maschine

Ergonomische Datendarstellung
• *Daniel Greitens*

Daten sind nur so gut, wie dem Betrachter verständlich. Dieser Vortrag liefert psychologische Hintergründe und einen strukturierten Weg, wie Daten optimal aufbereitet werden können.

Programmänderung vorbehalten

10110101110011010110
10110101110011011101101

Workshops: 19. April 2016

Smart Data vs. Big Data

Das Buzzword Big Data ist aktuell in aller Munde. Soziale Netzwerke, Sensoren, das Internet der Dinge und Server-Logs generieren massenhaft Daten in den verschiedensten Formaten. Doch es ist nicht damit getan, Daten zu sammeln. Der richtige Einsatz und eine smarte Verarbeitung machen aus Datenmengen erst Smart Data. Und genau hier liegt ein großes Potenzial: Unternehmen verzeichnen geringere Kosten und mehr Umsatz durch den Einsatz von Big-Data-Technologien. Diese bergen jedoch auch einige Herausforderungen. Datenschutz und -sicherheit sind nur ein Aspekt. Viele Unternehmen kämpfen noch mit mangelndem technischen und fachlichen Know-how.

Genau hier setzt die SMART DATA Developer Conference an. Sie bietet technisches und fachliches Know-how für Softwareentwickler, die einen entscheidenden Teil in Big Data Projekten beitragen.

Workshop 1: CQRS und Multi-Model-DB – ein Herz und eine Seele

Jan Fellien Uhrzeit: 09.00 – 18.00 Uhr

Lange habe ich nach einer guten Lösung für eine Datenbank gesucht, die sich ins CQRS Umfeld gut einfügt. Mit der OrientDB scheint die Suche ein Ende zu haben. Hiermit können Graphen und Documents gleichermaßen gespeichert (sprich ein Event Store) und die Read Models passgenau zu den Anforderungen abgelegt werden.

Der Workshop führt Sie zunächst in das CQRS-Paradigma ein und leitet Sie behutsam hinüber in die Datenspeicherung mit node.js und OrientDB.

Workshop 2: Smart Development mit Enapso

Alexander Schulze Uhrzeit: 09.00 – 18.00 Uhr

Big-Data-Analysen erzeugen Wissen als Basis für smarte Maschinen, für Vorhersagen, für Empfehlungs- und Assistenz-Systeme. Doch smarte Technologien verbessern nicht nur Software als Ergebnis, sondern auch deren Entwicklung.

Dieser Workshop stellt Modelle, Abfragen und Regeln semantischer Ontologien mit OWL2, SPARQL und SWRL vor. Wir erläutern Machine-Learning, BPMN mit Micro-Services in der Cloud und wie in Enapso intelligente Agenten Anforderungen und Wissen verbinden, als Assistenten zur Softwareentwicklung.

Geplant (lassen Sie sich vormerken):

Workshop 3: Smart Data with Microsoft Azure

Workshop 4: Eine smarte Architektur für Big Data

Die Referenten der SMART DATA Developer Conference (u.a.):

Jan Fellien,
devCrowd GmbH

Daniel Greitens,
MAXIMAGO GmbH

Werner Keil,
Creative Arts & Technologies

Constantin Klein,
Freudenberg IT SE & Co. KG

Olivia Klose,
Microsoft Deutschland GmbH

André Krämer,
Software, Training & Consulting

Jan-Hendrik Lendholt,
Otto (GmbH & Co KG)

Stefan Lieser,
CCD School

Michael Nolting,
Sevenval Technologies GmbH

Diego Poza,
Auth0 Inc.

Peter Rozek,
ecx.io germany GmbH

Alexander Schulze,
Innotrade GmbH

Rupert Steffner,
Otto (GmbH & Co KG)

Marcel Tilly,
Microsoft Research

Thomas Worm,
DATEV eG

SWIFT-PERFORMANCE OPTIMIEREN

Code optimieren

Auch Swift-Code kann in Sachen Performance noch weiter optimiert werden.

Apple stellt gerne die sehr gute Performance ihrer neuen Programmiersprache Swift als einen der Hauptvorteile im Vergleich zu Objective-C heraus. Das bezieht sich nicht nur auf Merkmale der Sprache selbst wie beispielsweise die durchgängige Typsicherheit, mit der in Swift beispielsweise immer klar und eindeutig definiert wird, welche Typen von Objekten ein Array enthalten darf.

Auch der Compiler wird in sehr großem Umfang von Apple gepflegt und stets optimiert, um auch das letzte Quäntchen Leistung aus dem eigenen Quellcode zu kitzeln. Viele dieser Optimierungen erhalten Entwickler bereits automatisch und »out of the Box«, indem sie einfach Swift in ihren App-Projekten einsetzen. An mancher Stelle können aber auch wir Entwickler nachhelfen und die Performance unseres Codes noch weiter optimieren.

Reference Counting

Jeder Apple-Entwickler dürfte mit der Technik des Reference Counting vertraut sein, ist es doch die Grundlage der Speicherverwaltung in der Entwicklung für OS X, iOS, watchOS und tvOS; ganz gleich ob man nun mit Objective-C oder Swift arbeitet. Bei diesem Referenzzählen werden ganz konkret die aktiven Verweise auf ein Objekt im Speicher gezählt. Gibt es keine Verweise mehr auf ein solches Objekt, wird es nicht mehr benötigt und der entsprechende Speicher freigegeben. Ein einfaches Beispiel dazu zeigt der nachfolgende Code:

```
var firstReference: MyClass? = MyClass()
var secondReference: MyClass? = firstReference
firstReference = nil
secondReference = nil
```

Während zu Beginn der OS X- und iOS-Entwicklung die Entwickler noch selbst dafür Sorge tragen mussten, den Referenzzähler mit Hilfe von *retain*- und *release*-Aufrufen auf dem aktuellen Stand zu halten, hat Apple diesen Aspekt bereits vor Jahren mit der Einführung des sogenannten Automatic Reference Counting (kurz ARC) deutlich vereinfacht.

Die *retain*- und *release*-Methoden waren damit Geschichte und der Compiler kümmert sich selbsttätig um das Ausführen der entsprechenden Methoden zum Erhöhen beziehungsweise Verringern eines Referenzzählers. So weit, so gut.

In Swift werden nur dann Referenzen auf ein Objekt angewendet, wenn es sich bei dem entsprechenden Objekt um eines vom Typ einer Klasse handelt. Klassen sind in Swift sogenannte Reference Types und werden daher über Verweise im Speicher referenziert. Bei Typen, die als Structure oder Enumeration definiert werden, handelt es sich um Value Types.

Objekte von Value Types werden immer kopiert, sobald sie einer anderen Variable oder Konstanten oder einer Funktion übergeben werden, wodurch niemals mehrere Referenzen auf ein und dasselbe Objekt verweisen. Dieses grundlegende Verhalten von Value Types und Reference Types in Swift ist ausschlaggebend für alle Optimierungen, die wir im Bereich des Reference Counting unserem Swift-Code angedeihen lassen können.

Klassen ohne Reference Types

Als Apple-Entwickler wissen wir: Wir arbeiten objektorientiert und damit primär mit Klassen. Die Frameworks von Apple bestehen hauptsächlich aus verschiedensten Klassen, die wir entweder direkt verwenden oder mittels eigener Subklassen um eigene Funktionen erweitern; der typische Alltag eines jeden Apple-Entwicklers.

Oftmals denkt man möglicherweise schon gar nicht mehr darüber nach, ob man einen bestimmten Typ nicht auch einfach als Structure statt als Klasse definieren könnte. Klassen sind nun einmal der hervorstechendste und die am häufigsten verwendete Art von Typ, wenn wir Apps für OS X, iOS, watchOS oder tvOS entwickeln.

Wie wir aber wissen, sind Klassen Reference Types. Die Verwendung einer Klasse bringt somit immer die Notwendigkeit der automatischen Speicherverwaltung mittels ARC mit sich. Auch wenn wir dazu selbst nicht direkt Code schreiben müssen, der sich um diese Speicherverwaltung kümmert, so findet diese intern doch jedes Mal statt. Und sie frisst Ressourcen. Ein kleines Beispiel soll das verdeutlichen.

Nehmen wir einmal eine Klasse, die selbst über keinerlei Reference Types verfügt, sondern nur Variablen und Konstanten in Form von Value Types definiert (zum Beispiel *Int, Float, String et cetera)*. Und nehmen wir weiterhin an, wir erstellen an einer anderen Stelle ein Array, welches Objekte eben jener Klasse enthält.

Zum Auslesen dieses Arrays kann dann die bekannte *for-in*-Schleife verwendet werden, um nach und nach auf alle Verweise der Objekte innerhalb des Arrays zuzugreifen. Listing 1 stellt diese Konstellation in vereinfachter Form beispielhaft dar.

Das ist Code, wie wir ihn in der Regel alle kennen und regelmäßig selbst schreiben und anwenden. Zu beachten ist dabei, das die erstellte Klasse *Person* nur Variablen in Form von Value Types definiert und somit nicht über Referenzen auf andere Objekte verfügt. Interessant ist, was nun innerhalb der *for-in*-Schleife passiert. Denn neben unserem Code, den wir innerhalb der Schleife ausführen lassen, werden im Hintergrund automatisch weitere Funktionen ausgeführt.

Ganz zu Beginn der Schleife, vor Ausführung des ersten Befehls, wird der Referenzzähler für das im jeweiligen Schleifendurchlauf aktuelle *person*-Objekt um eins erhöht; schließlich handelt es sich bei dieser Konstanten um einen neuen und weiteren Verweis auf das zugrunde liegende Objekt.

Ebenso wird immer am Ende der *for-in*-Schleife, nachdem unser letzter Befehl ausgeführt wurde, jener Referenzzähler wieder um eins verringert; schließlich muss dann der Verweis auf das entsprechende Objekt wieder freigegeben werden, da es innerhalb der *for-in*-Schleife nicht mehr benötigt wird. Und eben jener Prozess – Erhöhung des Referenzzählers zu Beginn und Verringerung am Ende – wiederholt sich bei jedem Schleifendurchlauf für jedes Objekt innerhalb des Arrays. Listing 2 stellt dieses Verhalten noch einmal detailliert dar.

Und dabei müsste das an dieser Stelle gar nicht sein. Einfacher und schneller wäre es, stattdessen direkt auf das eigentlich zugrunde liegende Objekt des Arrays zuzugreifen und dieses zu verwenden. Und tatsächlich lässt sich dieses Verhalten auch sehr einfach umsetzen, indem man den Typ *Person* nicht als Klasse, sondern als Structure definiert, wie in Listing 3 zu sehen.

Durch diese kleine Änderung am Code ist es an der besagten Stelle innerhalb der *for-in*-Schleife nun nicht länger nötig, das Reference Counting zu bemühen. Stattdessen haben wir direkten Zugriff auf den kopierten Value Type und sparen dem Programm Performance. Gerade bei sehr großen Arrays ist das durchaus eine spürbare und lohnenswerte Form der Code-Optimierung.

Allerdings gilt dabei zu beachten: Enthält die Structure Variablen oder Konstanten auf Basis eines Reference Types, so greift für diese auch automatisch wieder ARC und ein entsprechendes Referenzzählen findet statt. Den genannten Vorteil spielt man also typischerweise mit Structures aus, die nur über Value Types verfügen.

Structures und Reference Types

Betrachten wir in diesem Zusammenhang nun noch ein weiteres Beispiel, das eine mögliche Optimierung des eigenen Swift-Codes im Bereich des Reference Counting aufzeigt. Nehmen wir dazu dieses Mal eine Structure, die über viele verschiedene Variablen verfügt, die alle einem Reference Type entsprechen. Erstellen wir ein Objekt dieser Structure, so wird für all diese Variablen der Referenzzähler initial auf eins gesetzt. So weit, so gut.

Erstellen wir nun aber eine Kopie dieser Structure und weise diese einer anderen Variable per Zuweisung zu, so muss auch der Referenzzähler für jede der in der Structure existierenden Reference Types entsprechend erhöht werden. Wie zuvor beim Beispiel der *for-in*-Schleife kann das bei umfangreichen Structures mit vielen Reference Types durchaus Performance-Einbußen verursachen.

In solch einem Fall kann es sinnvoll sein, eine Art Wrapper-Klasse zu erstellen, die das entsprechende Structure enthält. Werden dann an anderer Stelle weitere Verweise auf diese Structure benötigt, so wird nicht die Structure selbst mit ihren vielen Reference Types kopiert, sondern stattdessen eine neue Referenz auf die Wrapper-Klasse erzeugt. Somit erhöht sich zwar deren Referenzzähler, die der vielen zugrunde liegenden Reference Types der Structure bleiben aber unverändert und müssen nicht erhöht oder angepasst werden.

Listing 1: Array mit Objekten einer Klasse auslesen

```
class Person {
  var firstName: String?
  var lastName: String?
}
var personArray: [Person] = ...
for person in personArray {
  ...
}
```

Listing 2: Array mit Verweis aufs Referenzzählen

```
class Person {
  var firstName: String?
  var lastName: String?
}
var personArray: [Person] = ...
for person in personArray {
  <Erhöhung des Referenzzählers für person>
  ...
  <Verringerung des Referenzzählers für person>
}
```

Listing 3: Array mit Objekten einer Structure auslesen

```
struct Person {
  var firstName: String?
  var lastName: String?
}
var personArray: [Person] = ...
for person in personArray {
  ...
}
```

Dynamic Dispatch

Ein mächtiges Sprach-Feature von Swift ist das sogenannte Dynamic Dispatch. Darüber prüft Swift, an welcher Stelle es eine bestimmte Funktion aufrufen oder einen spezifischen Wert auslesen soll, und spielt in diesem Zusammenhang bei Vererbung eine zentrale Rolle. Ein typisches Beispiel in diesem Zusammenhang zeigt Listing 4.

Es werden zwei Klassen *Vehicle* und *Motorboat* definiert, wobei Letztere eine Subklasse von *Vehicle* ist. Eine derartige Arbeit mit Klassen gehört zum Alltag eines jeden Apple-Entwicklers und ist unser täglich Brot. Und an dieser Stelle ha- ▶

ben wir die Möglichkeit, unseren Code mit entsprechenden Schlüsselwörtern weiter zu optimieren – sofern es die Programmlogik hergibt, doch dazu komme ich gleich.

Betrachten wir nun einmal, wie Dynamic Dispatch auf Grundlage des genannten Szenarios arbeitet, wenn wir an anderer Stelle in unserem Code eine Funktion wie in Listing 5 gezeigt haben, die als Parameter ein *Vehicle*-Objekt erwartet und darauf die Methode *startDriving* aufruft.

Nun, was geschieht in dieser Funktion? Es wird der Wert der Property *manufacturer* ausgegeben sowie die Methode *startDriving* auf dem übergebenen *Vehicle*-Objekt aufgerufen. Das ist das, was wir Entwickler implementiert haben und auch offenkundig sehen. Doch was verborgen bleibt, ist die interne Arbeit, die Swift zusätzlich noch zu durchlaufen hat, um diese Funktion erfolgreich ausführen zu können.

Denn das Problem ist: *Vehicle* könnte über Subklassen in unserem Programm verfügen, und eben jene Subklassen könnten eine eigene Implementierung für die Property *manufacturer* oder die Methode *startDriving* besitzen. In einem solchen Fall sorgt Dynamic Dispatch dafür, dass auch die korrekte Funktion des übergebenen Objekts aufgerufen wird.

Wie man sich vorstellen kann, verbraucht ein solcher Vorgang Ressourcen und Performance. Wenn es daher aufgrund der eigens definierten Klassenstruktur nicht sinnvoll ist, das bestimmte Funktionen einer Klasse von Subklassen überschrieben werden können, sollten diese entsprechend gekennzeichnet werden. In Swift gibt es dazu das Schlüsselwort *final*.

final wird vor die Deklaration einer Property oder Methode gesetzt, um dafür zu sorgen, dass diese nicht von Subklassen der entsprechenden Klasse mit einer eigenen Implementierung überschrieben werden können. Entsprechend weiß auch Swift, dass es in solch einem Fall für die entsprechenden Eigenschaften nicht nach passenden Implementierungen in möglichen Subklassen suchen muss.

Dadurch kann bereits mit dieser kleinen Änderung die Performance von Swift-Code deutlich verbessert werden. In unserem Beispiel aus Listing 5 könnten beispielsweise so die Property *manufacturer* und *speed* als *final* gekennzeichnet werden, da von unserer Seite nicht erwartet wird, dass Subklassen die Logik dieser Eigenschaften verändern. Listing 6 zeigt die entsprechende Aktualisierung der Klassen *Vehicle* und *Motorboat*.

Access Control

Eine weitere Möglichkeit zur Optimierung des eigenen Swift-Codes im Zusammenspiel mit Dynamic Dispatch stellen die Access Control Level dar, die in Swift zur Kennzeichnung von Klassen, Properties und Methoden verwendet werden können. Diese geben an, inwieweit die Eigenschaften einer Sourcecode-Datei oder eines Moduls in Swift auch in anderen Bereichen verfügbar sind. Insgesamt existieren in Swift drei Level, um die Access Control im Code abzubilden:

- *private:* Die Eigenschaft ist nur innerhalb der Sourcecode-Datei zugänglich, in der sie deklariert wurde.
- *internal:* Die entsprechende Eigenschaft steht innerhalb des Moduls zur Verfügung, in dem sie definiert wurde.

Listing 4: Dynamic DispatchStructure

```
class Vehicle {
  var manufacturer: String?
  var speed: Int?
  func startDriving() {
    speed = 10
  }
  func printVehicleInfo() {
    print("Hersteller: \(manufacturer)")
    print("Geschwindigkeit: \(speed)")
  }
}
class Motorboat: Vehicle {
  var boatType: String?
}
```

Listing 5: Funktion zum Starten eines Vehicle

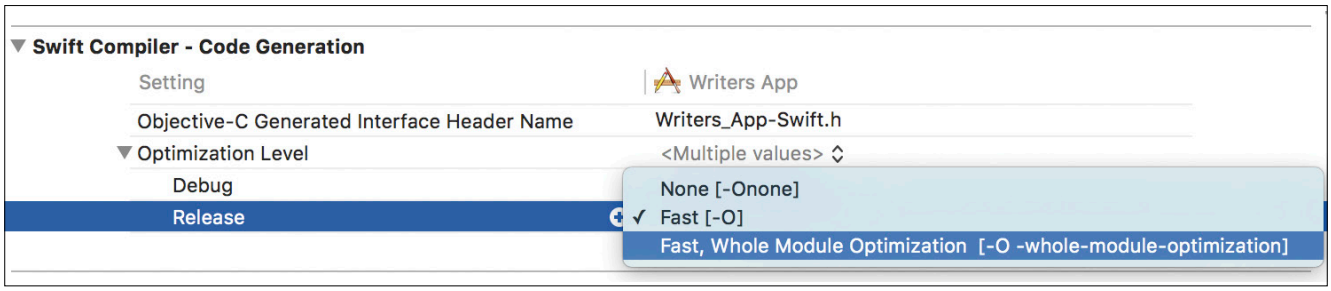
```
func startVehicle(vehicle: Vehicle) {
  print("Starte Fahrzeug von Hersteller \
  (vehicle.manufacturer).")
  vehicle.startDriving()
}
var myMotorboat = Motorboat()
myMotorboat.manufacturer = "Crownline"
startVehicle(myMotorboat)
```

Listing 6: Verwendung von final

```
class Vehicle {
  final var manufacturer: String?
  final var speed: Int?
  func startDriving() {
    speed = 10
  }
  func printVehicleInfo() {
    print("Hersteller: \(manufacturer)")
    print("Geschwindigkeit: \(speed)")
  }
}
class Motorboat: Vehicle {
  var boatType: String?
}
```

- *public:* Die entsprechende Eigenschaft steht immer zur Verfügung, auch von anderen Modulen aus.

Der Access Level *internal* stellt dabei den Standard dar und gilt immer, wenn kein anderer Level gesetzt ist. An vielen Stellen ergibt es aber beispielsweise Sinn, eine Property oder



Performance: Mittels Whole Module Optimization kann die Performance von Swift-Code deutlich verbessert werden (Bild 1)

Methode rein als *private* zu deklarieren und somit sicherzustellen, dass sie an keiner anderen Stelle im Programm zur Verfügung steht. Gerade für die internen Abläufe und Eigenschaften einer Klasse ist dieses Access Level sehr oft sinnvoll. Und setzt man es passend an den korrekten Stellen, verbessert man damit sogleich die Performance des eigenen Swift-Codes. Denn auch bei den Access Levels gilt: Wenn Swift weiß, dass eine Eigenschaft aufgrund der *private*-Deklaration nur innerhalb einer einzige Klasse implementiert sein kann und in keinerlei Subklassen oder anderen Modulen Verwendung findet, ist auch eine entsprechende Überprüfung vonseiten Swifts unnötig.

Whole Module Optimization

Neben den bisher gezeigten Optimierungen des eigentlichen Swift-Quellcodes gibt es noch eine weitere Möglichkeit, schnell und effizient den eigenen Swift-Code zu verbessern. Apple bezeichnet dieses Feature als Whole Module Optimization. Bevor man sich damit beschäftigt, was dieses Feature tut, und verstehen kann, wie es den eigenen Swift-Code optimiert, möchte ich zunächst das grundlegende Prinzip von Modulen und Source Files in Swift erläutern.

In Swift werden die Quellcode-Dateien in Modulen zusammengefasst. Ein Modul bildet dabei eine logische Gruppe für mehrere zusammenhängende Source Files, daher sind Module typischerweise Frameworks oder Apps.

Swift prüft bei Funktionsaufrufen und Eigenschaften immer ein zugehöriges Source File nach dem anderen. Dadurch kommt es auch zu den bereits im Zusammenspiel mit Dynamic Dispatch beschriebenen Problemen, dass Swift so beispielsweise bei Vererbung erst einmal prüfen muss, ob es nicht noch an irgendeiner anderen Stelle eine Subklasse mit einer eigenen Implementierung der gewünschten Funktionalität gibt. Hier fehlt eine übergreifende Modulsicht, die es Swift erlauben würde, selbst ohne Keywords wie *final* oder *private* zu erkennen, ob es eine alternative Implementierung für eine Eigenschaft gibt oder nicht. Und genau da kommt die Whole Module Optimization ins Spiel.

Links zum Thema

- Swift-Blog von Apple
  https://developer.apple.com/swift/blog

Bei Whole Module Optimization handelt es sich um ein neues Optimization Level, das Sie für Swift-Projekte in den Build Settings für den Compiler aktivieren können (Bild 1). Ist dieser Optimization Level gesetzt, setzt der Compiler zukünftig die Source Files eines Moduls vollständig zusammen und weiß so bei Funktionsaufrufen von vornherein, ob diese beispielsweise über eine zusätzliche Subklasse erfolgen oder nicht. Mit dieser für uns kleinen Änderung erhält Swift die Rundumsicht auf ein Modul und erkennt sofort, wie es bestimmte Eigenschaften aufzurufen und zu nutzen hat, ohne dafür diverse Prüfungen durchzuführen und damit die Performance in Mitleidenschaft zu ziehen.

Gerade aufgrund des geringen Aufwands für uns Entwickler, Whole Module Optimization in eigenen Projekten zu aktivieren, und des daraus resultierenden Performance-Gewinns sollte sich jeder einmal diesen neuen Optimization Level unbedingt näher ansehen.

Fazit

Swift ist von Haus aus eine sehr sichere und stabile Sprache. Nichtsdestoweniger sollte man an die in diesem Artikel vorgestellten Fallstricke im Blick behalten, um den eigenen Swift-Code optimieren zu können. Bereits für uns einfache Aktionen wie das klare Deklarieren von Properties und Methoden mittels *final* oder *private* hilft ungemein, die Performance der eigenen Apps zu verbessern. Und gerade große und komplexe Projekte können so von bisweilen sehr einfachen Optimierungen profitieren.

Nicht zuletzt sorgt die neue Whole Module Optimization dafür, mit einem Klick in den Build Settings die Performance unseres Swift-Codes deutlich zu verbessern. Hier zeigt es sich deutlich, mit wie viel Engagement und Herzblut Apple an Swift arbeitet. Freuen wir uns also auf die kommenden Optimierungen. ■

Thomas Sillmann
ist leidenschaftlicher 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 Kurzgeschichten.

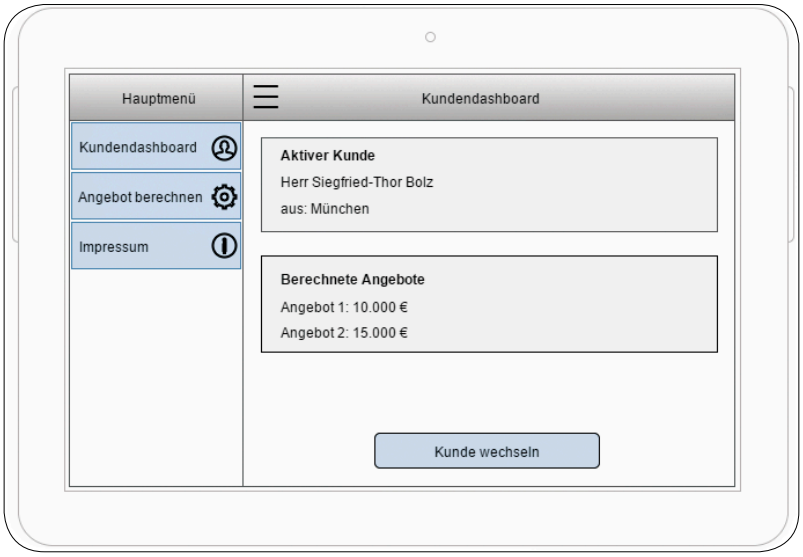
MODULARISIERTE APPS MIT GRUNT UND PHONEGAP

Modularisierung

Modularisierung ist bei einem umfangreichen Projekt das Gebot der Stunde.

Ich stieg im Frühjahr 2015 in ein bereits seit Herbst 2014 laufendes PhoneGap-Projekt ein. Es war ein technologischer Mix aus AngularJS, jQuery, Bootstrap, Ionic und anderen JavaScript-Frameworks. Der Code war ziemlich verworren, riesige Factory- und Controller-Dateien existierten, wovon eine davon sämtliche Abhängigkeiten zu beinhalten schien. Ein nettes Projekt, wie man es eben kennt, das schnell gewachsen ist, ein wenig nach dem Motto »functions first, code quality last«.

Es wurde für alle drei relevanten Tablet-Plattformen (iOS, Android, Windows 8) munter daran weiterentwickelt. Der Aufwand, diese eine App für alle drei Plattformen zur Verfügung zu stellen, war enorm hoch. Dann kam der Kunde auf die Idee, diese App für andere Märkte zur Verfügung zu stellen, allerdings jeweils mit einem anderen (abgespeckten) Funktionsumfang oder gar völlig neuen Funktionen, welche zugeschnitten wären für diesen neuen Markt (zum Beispiel ein anderer Rechenkern, Tracking, Farbgestaltung).

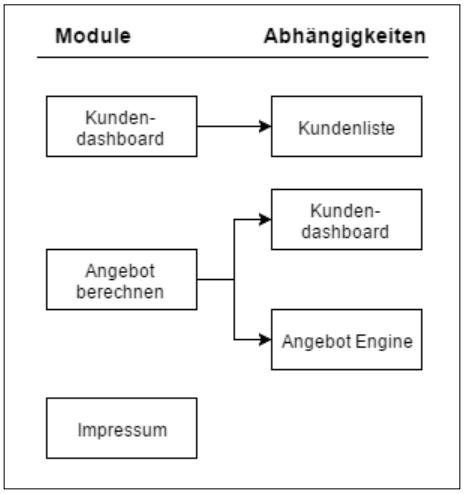


Entwurf der App: Hauptmenü mit diversen Buttons (Bild 1)

Wichtig war dabei, dass Funktionen, die der Markt niemals verwenden durfte (weil nicht bezahlt oder einfach nicht verfügbar) komplett aus dem Programmcode entfernt werden mussten. Ein einfaches Ausblenden im Code à la *ng-if='!marktDE'* war nicht ausreichend. Auf Grund von Reverse Engineering wäre es möglich gewesen, die deaktivierten Funktionen zu reaktivieren. Dies wollte man unbedingt vermeiden.

Eine Möglichkeit, diese Anforderungen abzubilden, wäre gewesen, einen neuen Branch im Repository anzulegen für den Markt X und irgendwann für den Markt Y und so fort. Hier den Code synchron zu halten ist schwierig. So etwas wollte ich nicht haben – die App-Modularisierung mit Grunt war geboren.

Diese Architektur-Beschreibung zielt primär auf existierende Projekte, funktioniert aber natürlich auch mit Projekten von der grünen Wiese. Wer das Glück hat, solch ein Projekt von Beginn an zu konzipieren, kann hiermit den Grundstein legen, um die App zukunftssicher zu gestalten. Alle anderen hingegen haben nun einen interessanten Weg vor sich, ihre App-Entwicklung modularer zu gestalten, denn die Märkte werden kommen.



Abhängigkeiten: Die ermittelten Modul-Abhängigkeiten für die App (Bild 2)

Eine einfache App modularisieren

Um zu verstehen, worum es hier geht, schauen Sie sich den sehr einfach gehaltenen Entwurf dieser App an (Bild 1). Zu erkennen sind ein Hauptmenü mit den Buttons *Kundendashboard, Angebot berechnen* und *Impressum*. Zu sehen ist das geöffnete Kundendashboard mit dem Button *Kunde wechseln* und einigen Informationen zum ausgewählten Kunden. Ich halte dieses Beispiel so einfach wie möglich, da die grundlegende Vorgehensweise zum Identifizieren und Ausgliedern von Modulen immer dieselbe ist. Zur Strukturierung der App sind einige vorbereitende Schritte erforderlich, bevor es an die Modularisierung geht.

Identifizieren der Module

Anhand dieses einfachen Beispiels ist es leicht, die einzelnen Module und deren Abhängigkeiten zu identifizieren. In diesem Fall hat das Modul *Kundendashboard* eine Abhängigkeit zum Modul *Kundenliste,* in der ein neuer Kunde ausgewählt werden kann. Eine App nur mit dem Modul *Kundendash-*

board und ohne das Modul *Kundenliste* zu erzeugen würde keinen Sinn ergeben.

Die Funktion *Angebot berechnen* funktioniert nur mit einem aktiven Kunden (Modul *Kundendashboard)* und verwendet eine spezielle Angebots-Engine zur Berechnung diverser Werte für exakt diesen Markt (DE).

Das Impressum ist unabhängig, zeigt aber je nach konfiguriertem Markt andere Einträge an.

Zu sehen sind die ermittelten Modul-Abhängigkeiten für diese hypothetische App in Bild 2. Auf den dafür notwendigen Programmcode geht dieser Artikel nicht ein, er dient nur dazu die Modularisierung anhand dieser einfach strukturierten App zu erläutern.

Identifizieren von globalen Frameworks

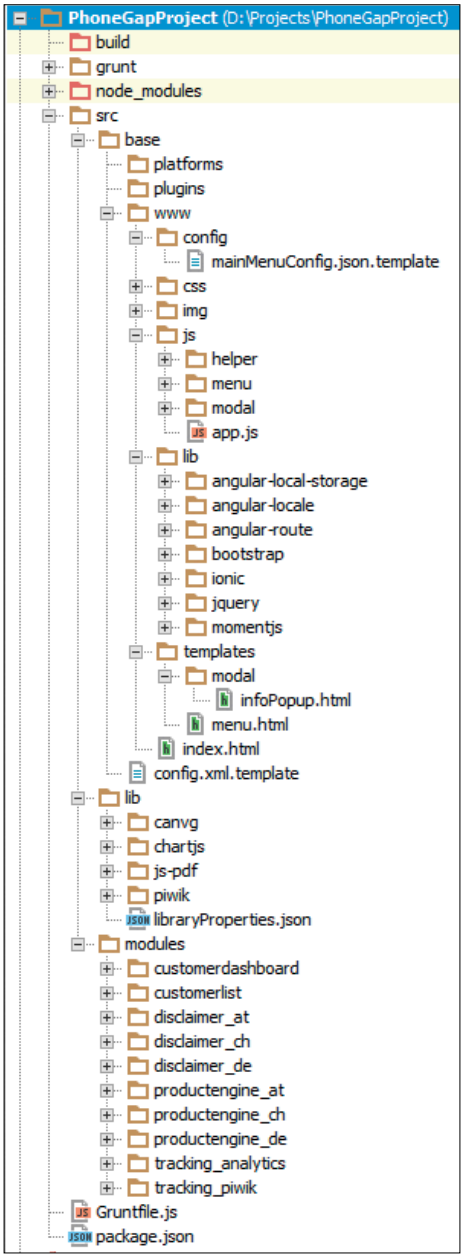
Das Entrümpeln der *index.html* wird ungern gemacht, muss aber sein. Frameworks wie AngularJS, Bootstrap und Ionic gehören in die Kategorie Global und werden immer geladen, Spezialfälle wie chartjs, js-pdf et cetera sollten nur geladen werden, wenn ein inkludiertes Modul diese benötigt. Ansonsten sind diese Frameworks nur Ballast und werden deshalb aus der *index.html* entfernt. Ein Platzhalter an der geeigneten Stelle in dieser Datei sorgt dafür, dass benötigte Module später an die richtige Stelle importiert werden können.

Aufbau der Projektstruktur

Die Verzeichnisstruktur dieser App in Bild 3 macht die Projektstruktur sichtbar. Im Verzeichnis *src* befinden sich die einzelnen App-Bausteine, die vom Build-Prozess zu einer funktionsfähigen App zusammengebaut werden. Im Verzeichnis *src/base/www* befindet sich die App noch im Rohzustand (ist dort nicht lauffähig) und es sind zwei Template-Dateien *(mainMenuConfig.json.template* und *config.xml.template)* vorhanden. Diese Templates enthalten Platzhalter, die erst im Build-Prozess mit gültigen Werten befüllt werden.

Das Verzeichnis *src/base* enthält den globalen Rumpfcode, der bei jedem App-Build verwendet wird. Dort sind auch die Dateien *index.html, app.js* (Startdatei der App) und die *config.xml.template* (PhoneGap-Konfigurationsdatei) enthalten.

Die globalen Frameworks (AngularJS, jQuery, Ionic …) befinden sich im Unterverzeichnis *lib,* und im Verzeichnis *templates* sind globale HTML-Dateien vorhanden, die immer benötigt werden – beispielsweise für das Hauptmenü und allgemeine Dialogfenster. In den Verzeichnissen *platforms* und *plugins* könnten spezifische Anpassungen hinterlegt werden, die beim Build Prozess verwendet werden. Diese werden in diesem Artikel allerdings nicht näher betrachtet.



Die Verzeichnisstruktur der App macht die Projektstruktur sichtbar (Bild 3)

Zu Beginn eines frischen Projekts könnte das *src/base/www*-Verzeichnis eventuell noch eigenständig lauffähig sein (zum Beispiel durch ein *ionic serve)*. Im Lauf der Zeit hingegen werden sicherlich einige Template-Dateien entstehen, deren Inhalte erst durch einen Build gesammelt und eingefügt werden.

In unserem Beispiel wäre dies die Datei *src/base/www/config/mainMenuconfig.json.template,* eine hypothetische Konfigurationsdatei, die als Inhalt ein JSON-Objekt enthält, das von einem *MenuController* ausgewertet werden kann.

Das Verzeichnis *src/lib* enthält JavaScript-Frameworks, die nur nach Bedarf hinzugezogen werden. Sofern ein Modul eine Abhängigkeit zu einem solchen Framework enthält, wird dieses geladen, ansonsten nicht. Die Datei *libraryProperties.json* beschreibt den Inhalt dieses Verzeichnis. Die benötigten Frameworks werden später zusammen mit den globalen Frameworks in ein entsprechendes Build-Verzeichnis kopiert und die *index.html* danach für diese Imports (JavaScript und CSS) angepasst.

Im Verzeichnis *src/modules* sind die einzelnen App-Funktionen als Module ausgelagert. Jedes Modul enthält sämtliche benötigten Dateien (JavaScript, CSS, HTML, JSON, Bilder …). Konfiguriert werden diese durch eine ebenfalls enthaltene *config.json*-Datei. Um eine lose Koppelung zu erreichen, gibt es hier keine Registrierungsdatei wie bei den JavaScript-Frameworks im Verzeichnis *src/lib.*

Aufbau eines Moduls

Bisher war viel von Modulen zu lesen, doch wie genau sieht so etwas nun aus? In Bild 4 ist der Verzeichnisinhalt des Moduls *Kundendashboard* zu sehen und in Listing 1 die dazugehörige Konfigurationsdatei.

Ein Modul enthält sämtliche benötigten Dateien, es gibt keine Abhängigkeiten zu Dateien, die sich in anderen Modulen (Verzeichnissen) befinden. Ein Modul kann aber in dessen Konfigurationsdatei angeben, dass es Abhängigkeiten ▶

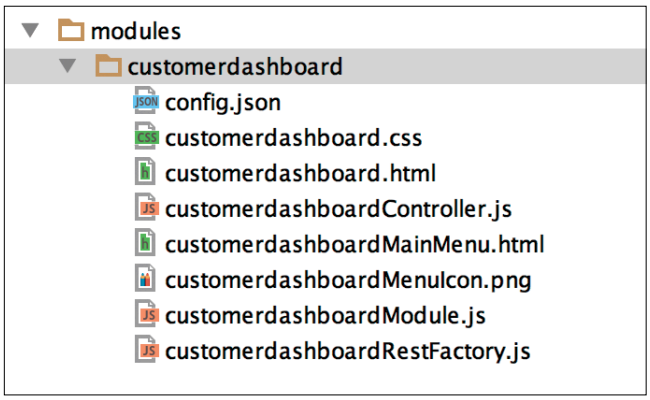
zu anderen Modulen und Frameworks hat (zum Beispiel benötigt das Modul *Kundendashboard* unbedingt das Modul *Kundenliste* und das fremde JavaScript-Framework jspdf).

Sogar das HTML zur Anzeige des Buttons *Kundendashboard* im Hauptmenü ist in einer separaten Datei *(customerdashboardMainMenu.html)* untergebracht und wird in der *config.json* beschrieben (Attribut *mainMenuButton)*.

Gehen wir nun auf die einzelnen Attribute in der *config.json* vom Kundendashboard ein.

Dies ist eine kleine Basis-Auswahl an Attributen, die zwingend benötigt werden, um ein Modul autark existieren zu lassen. Es steht dem Entwickler frei, hier eine Vielzahl an zusätzlichen Funktionsbeschreibungen unterzubringen. Jede davon muss dann von einem dafür angepassten Grunt-Task bearbeitet werden.

Das eindeutige Attribut *id* dient zum Identifizieren dieser Modul-Konfigurationsdatei. In der Projektdatei zu dieser App (Listing 2) werden sämtliche Module angegeben *(modules)*, die in der App verwendet werden sollen. Beim Build läuft dann ein spezieller ModuleManager-Task über dieses Daten-



Verzeichnisinhalt des Moduls Kundendashboard (Bild 4)

konstrukt und verwendet die ID als Verzeichnisnamen, um die jeweils dort enthaltende *config.json* zu laden *(src/modules/<id>/config.json)*. Aus diesem Grund ist die ID gleichbedeutend mit dem Verzeichnisnamen, da es keine Registrierungsdatei gibt, die alle Module umfasst. Dies wurde so konzipiert, um eine lose Koppelung zu erreichen.

Um eine vollständige Abkoppelung von den zu ladenden Angular-Modulen zu erreichen, ist es notwendig, diese Module aus der *app.js* fernzuhalten. In dem Attribut *appModuleInclude* wird daher das zu inkludierende Modul abgelegt. Ein Grunt-Task sammelt alle diese Informationen und fügt diese durch einen simplen Suchen-und-Ersetzen-Befehl in die Datei *app.js* ein. Zu sehen ist in Listing 3 der *modulemanagerAddAppModuleIncludes-Task* und in Listing 4 die sehr vereinfachte Datei *app.js* mit den Platzhaltern.

In den Attributen *js, css , images, json* und *view* werden die Dateien angegeben, die dieses Modul ausmachen. Diese werden (wie alle anderen Attribute auch) vom Task *modulemanagerCreateModuleData* in einer globalen Datenstruktur (im Gruntfile) gesammelt, um später von einem separaten

Listing 1: Konfigurationsdatei

```
{
  "id": "customerdashboard",
  "appModuleInclude": ["customerdashboard.module"],
  "js": ["customerdashboardController.js",
  "customerdashboardRestFactory.js",
  "customerdashboardModule.js"],
  "css": ["customerdashboard.css"],
  "images": ["customerdashboardMenuIcon.png"],
  "json": [],
  "view": ["customerdashboard.html",
  "customerdashboardMainMenu.html"],
  "mainMenuButton": {
    "id": "customerdashboard",
    "template": "templates/customerdashboard/
    customerdashboardMainMenu.html",
    "state": "app.customerdashboard",
    "tracking": {
      "id": 123
    }
  },
  "route": [{
    "state": "app.customerdashboard",
    "name": "customerdashboard",
    "url": "/customerdashboard",
    "cache": "true",
    "templateUrl": "templates/customerdashboard/
    customerdashboard.html",
    "controller": "customerdashboardController"
  }],
  "libDependencies" : ["jspdf"],
  "moduleDependencies": ["customerlist"]
}
```

Listing 2: Projektdatei

```
{
  "buildDirectory": "de",
  "bundleAppVersion": "1.0",
  "bundleAppName": "My App",
  "bundleIdentifier": "de.cqf.myapp",
  "bundleDescription": "App fuer den Markt DE",
  "bundleAuthorName": "Siegfried-Thor Bolz",
  "bundleAuthorEmail": "info@cq-factory.de",
  "bundleAuthorWebsite":
  "http://www.cq-factory.de/",
  "modules": ["customerdashboard", "customerlist",
  "disclaimer_de", "productengine_de",
  "tracking_analytics"],
  "mainMenuButtonModules": ["customerdashboard",
  "productengine_de", "disclaimer_de"],
  "libDependencies" : ["chartjs", "jspdf"]
}
```

Listing 3: modulemanagerAddAppModuleIncludes

```
module.exports = function (grunt) {
  grunt.registerTask
  'modulemanagerAddAppModuleIncludes',
  'Modify app.js to add the modules.', function () {
    var moduleData = grunt.config.get("moduleData");
    var sourceFile = "src/base/www/js/app.js";
    var targetFile = "build/de/www/js/app.js";
    var createdAppModuleString =
    createAppModuleIncludeString
    (moduleData.appModuleIncludes.
    getAllItemsAsList());
    grunt.task.run(["replacetextinfile:" + sourceFile
    + ":" + targetFile + "://@@appmodules@@:" +
    createdAppModuleString]);
  });

  function createAppModuleIncludeString(anArray) {
    var resultString = '';
    for (var i = 0; i < anArray.length; i++) {
      resultString += "\n, '" + anArray[i] + "'";
    }
    return resultString;
  }
};
module.exports = function (grunt) {
  grunt.registerTask('replacetextinfile', '', function
  (sourceFile, targetFile, stringToSearch,
  stringToReplace) {
    grunt.config.set('replace.source.file', sourceFile);
    grunt.config.set('replace.target.file', targetFile);
    grunt.config.set('replace.search.string',
    stringToSearch);
    grunt.config.set('replace.target.string',
    stringToReplace);
    grunt.task.run(['replace:file']);
  });
};

// Configuration for "grunt-text-replace"
module.exports = {
  file: {
    src: ['<%= replace.source.file %>'],
    dest: '<%= replace.target.file %>',
    replacements: [{
      from: '<%= replace.search.string %>',
      to: '<%= replace.target.string %>'
    }]
  },
  multiple: {
    src: ['<%= replace.source.file %>'],
    dest: '<%= replace.target.file %>',
    replacements: '<%= replace.replacements %>'
  }
};
```

Task in das Build-Zielverzeichnis kopiert zu werden. Der Sammelprozess ist in Listing 5 zu sehen. Selbstverständlich können hier noch weitere Datentypen untergebracht werden, die ein Modul benötigt. Das Objekt *mainMenuButton* beschreibt, wie der Button aussehen würde, wenn dieser im Hauptmenü gerendert werden soll. Angegeben ist das HTML-Template, der *state* und einige Tracking-Informationen. Der Aufbau dieses Objekts kann auch völlig anders gestaltet sein. Die Objekte *route* werden ebenfalls alle gesammelt, um später (wie beim *appModuleInclude)* in die *app.js* eingefügt zu werden. So ist gewährleistet, dass jedes Modul genau weiß, wie es aufgerufen werden kann.

Das Array *lib*Dependencies enthält IDs, die in der *library-Properties.json* (Verzeichnis *src/lib*) vorhanden sind. Zu sehen ist diese Konfigurationsdatei in Listing 6. In unserem Beispiel hat das Kundendashboard eine Abhängigkeit zum ▶

Listing 4: app.js

```
angular.module('myapp', ['ionic',
'LocalStorageModule', 'ngCordova',
'myapp.common.module'
// Placeholder for the GRUNT T
// ask modulemanagerAddAppModuleIncludes
//@@appmodules@@
])

  .config(function ($stateProvider) {
    $stateProvider
      .state('app', {
        name: "app",
        url: "/app",
        abstract: true,
        templateUrl: "templates/menu.html",
        controller: 'menuController'
      })

      // Placeholder for the GRUNT Task
      // modulemanagerAddRoutes
      //@@appRoutes@@
    ;
  }
);
```

Listing 5: Sammelprozess

```
module.exports = function (grunt)
{
  grunt.registerTask
  ('modulemanagerCreateModuleData',
  'Manages multiple module config data.',
  function ()
  {
    // Get access to the global moduleData.
    var moduleData = grunt.config.get("moduleData");
    var moduleConfigs =
    grunt.option('moduleConfigs');

    for (var index in moduleConfigs) {
      var moduleConfig = moduleConfigs[index];
      grunt.log.writeln("Reading module config
      [" + moduleConfig.id + "].");

      var moduleDirectory = moduleConfig.id;

      if (moduleConfig.appModuleIncludes) {
        addArrayContentToHashmap
        (moduleConfig.appModuleInclude,
        moduleData, "appModuleInclude",
        moduleDirectory);
      }

      if (moduleConfig.js) {
        addFilesToArray(moduleConfig.js,
        moduleData, "js", moduleDirectory);
      }
    }

    grunt.config.set("moduleData", moduleData);
  });

  function addFilesToArray(anArrayOfFiles,
  aModuleData, aProperty, moduleDirectory) {
    for (var i = 0; i < anArrayOfFiles.length; i++) {
      var fileObject = {};
      fileObject.sourceFile = moduleDirectory + "/"
      + anArrayOfFiles[i];
      fileObject.targetFile = moduleDirectory + "/"
      + anArrayOfFiles[i];
      aModuleData[aProperty].push(fileObject);
    }
  }

  function addArrayContentToHashmap(anArray,
  aModuleData, aProperty, id) {
    for (var i = 0; i < anArray.length; i++) {
      aModuleData[aProperty].put(id, anArray[i]);
    }
  }
};
```

Framework jspdf. Diese Abhängigkeiten werden alle von einem Grunt-Task gesammelt und später als Imports in die *index.html* hinzugefügt. Das Attribut *moduleDependencies* ist die kleinere Version der Modul-Abhängigkeiten von der Projektdatei. Hier kann auf Modul-Ebene angegeben werden, welche Module benötigt werden, damit dieses Modul funktionieren kann. Ein Plausibilitätscheck in Grunt kann dabei überprüfen, ob die gewünschten Module überhaupt von der Projektdatei geladen werden. Wenn nicht, dann muss das geforderte Modul in der Projektdatei hinzugefügt werden.

Die Projektdatei

Jede App-Variante benötigt eine eigene Projektdatei. Unser kleines Beispiel enthält nur die notwendigsten Einträge für den Markt DE. Diese wären:

- Verzeichnisname, der im Build-Ordner *(buildDirectory)* erzeugt wird.
- Zu verwendende Module *(modules*-Array).
- Bundle-Informationen, die in der *config.xml* von PhoneGap eingefügt werden können.
- Zu ladende Frameworks *(libDependencies)*.
- Die im Hauptmenü sichtbaren Buttons *(mainMenuButtonModules)*.

Beim Ausführen des Grunt-Tasks *buildProject* muss die Projektdatei als Parameter mit übergeben werden:

```
grunt buildProject -projectData=projectDE.json
```

Das Gruntfile ist für diesen Artikel sehr einfach aufgebaut (Listing 7). Das *Data*-Objekt speichert die mittels Parameter übergebene Projektdatei im Attribut *projectData*. Die Registrierungen der JavaScript-Frameworks landen im Attribut

Listing 6: libraryProperties.json

```
{
  "chartjs": {
    "css": [],
    "js": ["chartjs/js/chart.js"]
  },
  "jspdf": {
    "css": [],
    "js": ["js-pdf/js/jspdf.min.js"]
  },
  "piwik": {
    "css": [],
    "js": ["piwik/js/piwik.js"]
  },
  "canvg": {
    "css": [],
    "js": ["canvg/js/canvg.js",
    "canvg/js/rgbcolor.js", "canvg/js/StackBlur.js"]
  }
}
```

libraryData. Interessant ist hier der Blick auf das Konstrukt *moduleData*, in dem sämtliche Informationen aus allen verwendeten Modul-Konfigurationsdateien gespeichert werden. Dies geschieht während der Ausführung diverser *moduleManager*-Tasks. Um mehrfache Einträge zu vermeiden, wurde bei einigen Speichern eine HashMap-Datenstruktur verwendet (Listing 8).

Nachdem sämtliche Module identifiziert und extrahiert wurden, muss ein Grunt-Prozess (nennen wir diesen *buildProject)* etabliert werden, der alles zusammenbaut, abhängig von der konfigurierten Projektdatei (Listing 9). Dazu wird die Projektdatei eingelesen (geschieht im Gruntfile) und durch ▶

Listing 7: Gruntfile

```
module.exports = function (grunt) {

  eval(grunt.file.read('grunt/helper/HashMap.js'));
  var path = require('path');
  require('load-grunt-tasks')(grunt);
  grunt.loadTasks('grunt/tasks');

  require('load-grunt-config')(grunt, {
    configPath: path.join(process.cwd(),
    'grunt/configs'),
    init: true,
    data: {
      projectData: grunt.file.readJSON('grunt/
      projects/' + grunt.option('projectData')),
      libraryData: grunt.file.readJSON
      ('src/lib/libraryProperties.json'),
      moduleData: {
        loadedModules: new HashMap(),
        appModuleIncludes: new HashMap(),
        js: [],
        css: [],
        view: [],
        images: [],
        json: [],
        routes: new HashMap(),
        libDependencies: new HashMap(),
        moduleDependencies: new HashMap(),
        mainMenuButtonModules: new HashMap()
      }
    },
    mergeFunction: require('recursive-merge'),
    loadGruntTasks: {
      pattern: 'grunt-*',
      config: require('./package.json'),
      scope: 'devDependencies'
    },
    postProcess: function (config) {
    },
    preMerge: function (config, data) {
    }
  });
};
```

Listing 8: HashMap.js

```
HashMap = function () {
  this._dict = [];
};

HashMap.prototype._get = function (key) {
  for (var i = 0, couplet; couplet = this._dict[i];
  i++) {
    if (couplet[0] === key) {
      return couplet;
    }
  }
};

HashMap.prototype.put = function (key, value) {
  var couplet = this._get(key);
  if (couplet) {
    couplet[1] = value;
  } else {
    this._dict.push([key, value]);
  }
  return this; // for chaining
};

HashMap.prototype.get = function (key) {
  var couplet = this._get(key);
  if (couplet) {
    return couplet[1];
  }
};

HashMap.prototype.length = function () {
  return this._dict.length;
};

HashMap.prototype.getAllItemsAsList = function () {
  var retList = [];
  for (var i = 0; i < this._dict.length; i++) {
    retList[i] = this._dict[i][1];
  }
  return retList;
};

HashMap.prototype.remove = function (key) {
  for (var i = 0, couplet; couplet = this._dict[i];
  i++) {
    if (couplet[0] === key) {
      this._dict.splice(i, 1);
    }
  }
};
```

die nachfolgenden Tasks ausgewertet. Dabei werden die darin gespeicherten Informationen in temporäre Datenspeicher übernommen; diese befinden sich im Datenkonstrukt *moduleData* vom Gruntfile. Dies wären dann sämtliche geladenen Module (in *loadedModules),* die benötigten JavaScript-Frameworks *(libDependencies),* sämtliche Dateien und Konfigurationen aller Module, darunter auch die Buttons, die im Hauptmenü angezeigt werden sollen *(mainMenuButtonModules),* und die AngularJS-Module *(appModuleIncludes).*

Selbstverständlich können hier noch eine Vielzahl an zusätzlichen Informationen mit übernommen werden.

Der Standard-Prozess durchläuft dabei mehrere Schritte (Grunt Tasks), welche problemlos für Spezialanpassungen erweitert werden können. Hier werden nun die wichtigsten davon erläutert.

Der erste Task von *buildProject* lädt nun alle benötigten Modul-Konfigurationen (Task *loadModules).* Dabei werden nur die jeweiligen *config.json*-Dateien aus den entsprechenden Modul-Verzeichnissen *(src/modules/<id>/config.json)* geladen und in *moduleData .loadedModules* gespeichert. Sollte aufgrund einer riesigen Modulliste das ein oder andere Modul mehrfach angegeben worden sein, so ist dies kein Problem, da die interne Datenstruktur eine HashMap abbildet und als *key* die ID des Moduls verwendet und als Value den Inhalt der geladenen *config.json*. Dasselbe Prinzip gilt auch für die anderen Daten, die in einer HashMap landen.

Aus diesem Grund ist die *id* in der *config.json* gleichbedeutend mit dem Verzeichnisnamen in *src/modules/*.

Listing 9: buildProject.js

```
module.exports = function (grunt) {
  grunt.registerTask('buildProject', 'starting point
  for building your app.', function () {
    // Run tasks synchronously, in order.
    grunt.task
      .run("loadModules")
      .then(function () {
      })
      .run("moduleManager")
      .then(function () {
      })
      .run("menuManager")
      .then(function () {
      })
      .run("configxmlManager")
      .then(function () {
      })
      .run("doWhatEverManager")
      .then(function () {
      })
      .run("annotateManager")
      .then(function () {
      })
      .run("uglifyManager")
      .then(function () {
      })
      .run("zipManager")
      .then(function () {
        grunt.log.write('PRODUCTION FILE CREATED!').
        ok();
      })
    ;
  });
};
```

Module auswerten

Dieser übergeordnete Task (hier *moduleManager* genannt) iteriert dabei über sämtliche geladenen Modul-Konfigurationen *(moduleData .loadedModules)* und speichert die darin enthaltenen Werte in die entsprechenden Speicher des Konstrukts *moduleData* im Gruntfile. Dieser Task besteht für gewöhnlich aus mehreren Unter-Tasks wie zum Beispiel *modulemanagerCreateModuleData*.

Es wurden nun alle Daten gesammelt, welche diese App ausmachen. In den folgenden Schritten werden diese Daten ausgewertet.

Jedes Modul kann in seiner *config.json* angeben, welche Module unbedingt geladen sein müssen, um korrekt zu funktionieren. Im Fall eines Kundendashboard-Moduls wäre es schön, wenn ein Kundenliste-Modul zur Auswahl eines Kunden verfügbar wäre. Andernfalls würde es keinen Sinn ergeben. Welche Module auf jeden Fall geladen worden sind, steht in *loadedModules*, dessen Inhalte aus der eingelesenen Projektdatei stammen.

Zur Lösung des Problems kann man ganz einfach über das Attribut *moduleDependencies* einer jeden Konfigurationsdatei iterieren und prüfen, ob Modul X in *loadedModules* vorhanden ist. Durch die verwendete HashMap geht das recht angenehm mit der *get()*-Funktion.

Eine zusätzliche Überprüfungsmöglichkeit wäre ein sogenannter Modul-Ausschluss. Wenn es zum Beispiel mehrere Tracking-Module gibt, aber nur eines davon geladen sein darf, dann wäre in den betroffenen *config.json* Dateien ein Attribut *moduleExclusions* (Listing 10) sinnvoll, das eine Liste

Listing 10: config.json

```
{
  "id": "tracking_analytics",
  "appModuleInclude": ["tracking.module"],
  "js": ["tracking_analyticsModule.js",
  "tracking_analyticsFactory.js"],
  "css": [],
  "view": [],
  "route": [],
  "libDependencies" : [],
  "moduleDependencies": [],
  "moduleExclusions": ["tracking_piwik"]
}
```

an Modulen enthält, die nicht geladen werden dürfen. Zweckmäßig ist das unter anderem, wenn mehrere Tracking-Module zur Verfügung stehen (zum Beispiel weil jeder Markt eine andere Lösung bevorzugt), aber nur eines davon in die App inkludiert werden darf.

Modul-Dateien kopieren

Nachdem sämtliche Modul-Konfigurationen überprüft und validiert worden sind, ist es an der Zeit, die gesammelten Dateien (JavaScript, CSS, HTML, JSON, Bilder et cetera) an die richtigen Stellen im Build-Verzeichnis zu kopieren.

Da die ID einer Konfigurationsdatei gleichbedeutend ist mit dem Verzeichnisnamen, in dem sich dieses Modul befindet, kann der Kopierbefehl recht leicht erzeugt werden. Im Modul-Verzeichnis befinden sich alle Dateien eines Moduls an derselben Stelle, in der App dagegen müssen diese an die entsprechenden Stellen im Code-Baum kopiert werden:

- **JavaScript:** /build/de/www/js/<modulname>/
- **CSS:** /build/de/www/css/<modulname>/
- **HTML:** /build/de/www/templates/<modulname>/
- **Bilder:** /build/de/www/templates/<modulname>/

Zu sehen ist in Listing 11 der entsprechende Task *modulemanagerCopyModuleDataFiles* mitsamt der *copyFilesSync*-Funktion. Um einen synchronen Ablauf der Kopieroperationen zu gewährleisten, wurde hier das Grunt-Modul *glob-cp* verwendet. Dies ist notwendig, um im Anschluss daran notwendige Anpassungen in den kopierten Dateien vornehmen zu können (wenn notwendig!) und um den asynchronen Ablauf von Grunt zu ordnen.

Im Anschluss an das Kopieren muss nun die *index.html* angepasst werden, unter anderem mit dem Task *modulemanagerAddCSSImportsToIndexHtml* (Listing 12). Es ist notwendig, die CSS- und JavaScript-Imports einzubauen. In diesem Beispiel sehen Sie exemplarisch, wie dies für CSS-Imports mit ▶

Listing 11: modulemanagerCopyModuleDataFiles

```
module.exports = function (grunt) {
  grunt.registerTask
  ('modulemanagerCopyModuleDataFiles', 'copy all the
  collected files from the moduleData.', function ()
  {
    var moduleData = grunt.config.get("moduleData");
    var projectData =
    grunt.config.get('projectData');
    var targetDirectory = 'build/' +
    projectData.buildDirectory + '/';

    copyFilesSync(moduleData.js, targetDirectory +
    'www/js');
    // Copy all CSS files
    copyFilesSync(moduleData.css, targetDirectory +
    'www/css');
    // Copy all View files
    copyFilesSync(moduleData.view, targetDirectory +
    'www/templates');
  });

  function copyFilesSync(anArrayOfFiles,
  targetDirectory) {
    // https://www.npmjs.com/package/glob-cp
    var cp = require('glob-cp');
    for (var i = 0; i < anArrayOfFiles.length; i++) {
      cp.sync('src/modules/' +
      anArrayOfFiles[i].sourceFile, targetDirectory
      + '/' + anArrayOfFiles[i].targetFile);
    }
  }

};
```

Listing 12: modulemanagerAddCSSImportsToIndexHtml

```
module.exports = function (grunt)
{

  grunt.registerTask('modulemanagerAddCSSImportsTo
  IndexHtml',   'Modify index.html and add css
  imports.', function () {
    var projectData = grunt.config.
    get('projectData');
    var moduleData = grunt.config.get("moduleData");
    var targetFile = 'build/' +
    projectData.buildDirectory + '/www/index.html';
    var createdCSSString =
    createCSSIncludes(moduleData.css);

    grunt.config.set('replace.source.file',
    targetFile);
    grunt.config.set('replace.target.file',
    targetFile);
    grunt.config.set('replace.search.string',
    "<!--@@cssModuleIncludes@@-->");
    grunt.config.set('replace.target.string',
    createdCSSString);
    grunt.task.run(['replace:file']);
  });

  function createCSSIncludes(anArray) {
    var resultString = '';
    var template = "  <link href='css/%1'
    rel='stylesheet'>\n";
    for (var i = 0; i < anArray.length; i++) {
      resultString += template.replace('%1',
      anArray[i].targetFile);
    }
    return resultString;
  }
};
```

dem Grunt-Modul *grunt-text-replace* erledigt werden kann (dies erfolgt analog für JavaScript-Imports). Dabei wird in der *index.html* nach dem String *<!--@@cssModuleIncludes@@-->* gesucht und dieser durch die CSS-Importe ersetzt. Dieses Modul kann bedarfsweise auch für mehrfache Ersetzungen verwendet werden.

In dem Attribut *appModuleInclude* der Modul-Konfigurationen befinden sich die in die *app.js* einzubindenden Module und im Attribut *route* die dazugehörigen Routen. Diese Daten wurden ebenfalls zuvor gesammelt und können nun zusammen mit dem Grunt-Modul *grunt-text-replace* in die *app.js* eingefügt werden.

Listing 13: index.html

```
<!DOCTYPE html>
<html>
  <head>
    <link href="lib/ionic/css/ionic.css"
    rel="stylesheet">
    <link href="lib/bootstrap/css/bootstrap.min.css"
    rel="stylesheet">
    <link href="css/ourStyle.css" rel="stylesheet">
    <!--@@cssLibDependencyIncludes@@-->
    <!--@@cssModuleIncludes@@-->
  </head>
  <body ng-app="yourApp">
    <script type="text/javascript"
    src="lib/jquery/js/jquery.min.js"></script>
    <script type="text/javascript"
    src="lib/bootstrap/js/bootstrap.min.js"></script>
    <script type="text/javascript"
    src="lib/ionic/js/ionic.bundle.js"></script>
    <script type="text/javascript"
    src="lib/momentjs/js/moment-locales.js"></script>
    <script type="text/javascript"
    src="lib/angular-route/js/angular-route.min.js">
    </script>
    <script type="text/javascript"
    src="cordova.js"></script>

    <!--@@jsLibDependencyIncludes@@-->

    <script type="text/javascript"
    src="js/helper/HashMap.js"></script>
    <script type="text/javascript"
    src="js/modal/modalFactory.js"></script>
    <script type="text/javascript"
    src="js/modal/modalModule.js"></script>

    <!--@@jsModuleIncludes@@-->
    <script type="text/javascript"
    src="js/global/app.js"></script>
  </body>
</html>
```

Was an Dateien noch fehlt, sind die auf Anfrage benötigten JavaScript-Frameworks wie chartjs, jspdf et cetera. Im Attribut *libDependencies* der Modul-Konfigurationen befinden sich die gewünschten Frameworks, die ebenfalls gesammelt worden sind. Hier gibt es einen kleinen Unterschied zum Vorgehen bei den Modulen.

Ein solches Framework besteht oft aus mehreren JavaScript- und CSS-Dateien, die ihre eigenen Unterverzeichnisse haben – mit eingebauten relativen Pfaden und anderen unschönen Dingen. Um nun das aufwendige Zerpflücken der einzelnen Dateien zu vermeiden, wurde als Lösung die Registrierungsdatei *libraryProperties.json* eingeführt.

Abhängige JavaScript-Frameworks kopieren

Dort werden sämtliche Dateien mit relativen Pfaden ausgehend vom jeweiligen Framework-Verzeichnis angegeben. Ein Grunt-Task iteriert nun über die gesammelten *libDependencies (moduleData.libDependencies),* und anhand der dort gespeicherten *id* kann der Iterator auf das Attribut des Ob-

Listing 14: mainMenuConfig.json

```
{
  "templates": [

    {
      "id": "customerdashboard",
      "template": "templates/customerdashboard/
      customerdashboardMainMenu.html",
      "state": "app.customerdashboard",
      "tracking": {
        "key": 123
      }
    },

    {
      "id": "productengine",
      "template": "templates/productengine_de/
      productengineMainMenu.html",
      "state": "app.productengine",
      "tracking": {
        "key": 456
      }
    },
    {
      "id": "disclaimer",
      "template": "templates/disclaimer_de/
      disclaimerMainMenu.html",
      "state": "app.disclaimer",
      "tracking": {
        "key": 789
      }
    }
  ]
}
```

jekts in der *libraryProperties.json* zugreifen und hat sämtliche benötigten Daten, um einen Kopiervorgang einzuleiten. Zusätzlich muss noch die *index.html* mit den neuen Imports angepasst werden.

In diesem Task werden sämtliche globalen Dateien aus dem Verzeichnis */src/base/www/* in das Build-Verzeichnis kopiert. Enthalten sind dabei auch die globalen Frameworks (AngularJS, Ionic, jQuery …), die ein Must-have sind. Diese zu modularisieren ergibt keinen Sinn. Die Imports dafür müssen nicht angepasst werden, sie stehen schon von Anfang an in der *index.html* (Listing 13).

Hauptmenü erzeugen

Das Attribut *mainMenuButtonModules* der Projekt-Konfigurationsdatei ist eine Map, welche als Key eine Modul-ID verwendet und als Value das Objekt *mainMenuButton* aus der entsprechenden Modul Konfigurationsdatei *(config.json)* enthält.

Diese Informationen wurden gesammelt und können nun zu einem gigantischen JSON-Objekt-String zusammengebaut werden, der einen Platzhalter in der Konfigurationsdatei *src/base/www/config/mainMenuConfig.json.template* ersetzt:

```
{
  "templates":
  @@mainMenuConfigInclude@@
}
```

Diese angepasste Datei (Listing 14) wird dann an die entsprechende Stelle im Build-Verzeichnis kopiert und kann vom Hauptmenü-Controller geladen und verwendet werden, um die Buttons für die View zu rendern und die Klicks darauf handzuhaben:

```
<!-- MAIN MENU -->
<ion-content>
  <!-- Use the mainMenuConfig.json for the button
  rendering. -->
  <div ng-repeat="item in model.mainMenuData.templates">
    <ng-include src="item.template"></ng-include>
  </div>
</ion-content>
```

Dafür verantwortlich wäre der Grunt-Task *menuManager*.

Dies wurde jetzt nur exemplarisch skizziert. Ein Modul kann selbstverständlich ebenfalls ein eigenes Menü enthalten, und indem man diesem ein Attribut wie *ownMenuButtons* gibt, kann ein Task dafür generisch Menü-Konfigurationen erzeugen, analog zum Task *menuManager*.

PhoneGap-Konfigurationsdatei anpassen

Diese Konfigurationsdatei liegt ebenfalls in einem Template-Format vor:

```
<?xml version='1.0' encoding='utf-8'?>
<widget id="@@widgetid@@" version="@@appversion@@"
xmlns="http://www.w3.org/ns/widgets"
  xmlns:cdv="http://cordova.apache.org/ns/1.0">

  <name>@@name@@</name>
  <description>
    @@description@@
  </description>
  <author email="@@authoremail@@" href="@@authorweb@@">
    @@authorname@@
  </author>
  <content src="index.html" />

</widget>
```

Die Werte für die Platzhalter sind in der Projektdatei enthalten. Der dazugehörige Task *configxmlManager* sucht die Platzhalter, ersetzt diese mit den Werten und kopiert anschließend die Datei in das Build-Verzeichnis.

Nachdem nun alle benötigten Dateien kopiert und die entsprechenden Dateien (*index.html, app.js* und diverse Konfigurationsdateien) angepasst worden sind, können nachfolgende Tasks vorbereitende Tätigkeiten für das Deployment vornehmen.

Dazu gehören Uglify (versucht JavaScript und CSS Dateien unlesbar zu machen), Concat (alle unlesbar gemachten Dateien in eine einzige Datei zusammenfügen) und Zippen (das komplette */build/de/www/*-Verzeichnis in ein Zipfile packen). Anschließend kann das Zipfile auf den Build-Server kopiert werden.

Fazit

In einem umfangreichen Projekt ist der Prozess der Modularisierung nicht von heute auf morgen zu erledigen. Es fängt damit an, dass neue Funktionen in dieser vorgestellten modularen Bauweise erzeugt werden müssen. Bei jeder Refactoring-Runde werden dann immer mehr Funktionen modularisiert. Zuerst die groben Hauptfunktionen (wie das Kundendashboard aus diesem Beispiel), und dann die darin enthaltenen Unterfunktionen.

Wichtig ist dabei, darauf zu achten, dass die Abhängigkeiten der Module untereinander nicht verloren gehen. Hält man sich strikt an die gezeigte Vorgehensweise, dann ist die App nicht nur leichter wartbar und für neue Team-Mitglieder leichter zu verstehen, sondern es erleichtert auch das Austauschen von Funktionen und das Erzeugen von Variationen für die Bedürfnisse anderer Märkte. ■

Siegfried-Thor Bolz
ist Diplom-Informatiker (FH) und als freiberuflicher IT-Berater mit den Schwerpunkten Adobe AEM und Mobile-Hybrid-App-Entwicklung für die CQ-Factory GmbH tätig. Sein Fokus liegt dabei auf der Interoperabilität beider Welten.
www.siegfried-bolz.de

XTEXT, XTEND: DOMAIN-SPECIFIC LANGUAGE FÜR C++ / QT (TEIL 3)

Qt Properties und Listen

Stressfreie Implementierung dank des Einsatzes einer DSL (Domain-Specific Language).

Die Grundzüge einer DSL haben wir im ersten Teil dieser Artikelserie kennengelernt, und der zweite Teil betrachtete immer wiederkehrende Muster in mobilen Businessanwendungen. Jetzt gehen wir wieder einen Schritt zurück und betrachten die Properties eines Datenobjekts im Detail.

Die Eigenschaften der Datenobjekte ändern sich durch Eingaben am mobilen Gerät oder den Abruf vom Server. Eine der immer wiederkehrenden Aufgaben ist es, diese Änderungen im UI sichtbar zu machen. Das heißt, wir benötigen einen Mechanismus, der entsprechende Events auslöst und verarbeitet. Bei Qt sind das die Signals und Slots.

Signals und Slots

Betrachten wir zunächst das Signals-und-Slots-Konzept allgemein. Ein Signal ist ein Funktionsaufruf, der dann erfolgt, wenn ein Ereignis (Event) oder eine Aktion erfolgt ist. Das Signal ruft alle Event Handler auf, die auf exakt dieses Signal lauschen.

Ein Slot kann mit einem Event Handler verglichen werden und wird von Signalen aufgerufen.

Ein Button beispielsweise hat ein *clicked()*-Signal, und eine Anwendung kann einen Slot implementieren, der exakt auf dieses Signal achtet. Neben den Standardsignalen wie *clicked()* bei einem Button kann eine Anwendung eigene Signale definieren und diese dann feuern, wenn erforderlich. Der dazu erforderliche Befehl lautet *emit <signal>*.

In der Header-Datei definieren wir eigene Signale wie folgt:

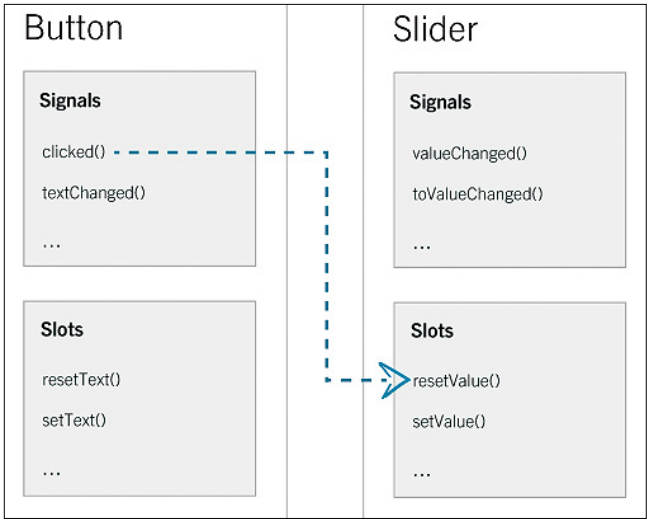
```
class MyObject: public QObject{
  Q_OBJECT
  ...
signals:
  void mySignal();
  ...
}
```

Und um das Signal abzufeuern, können wir an beliebiger Stelle *emit* ausführen:

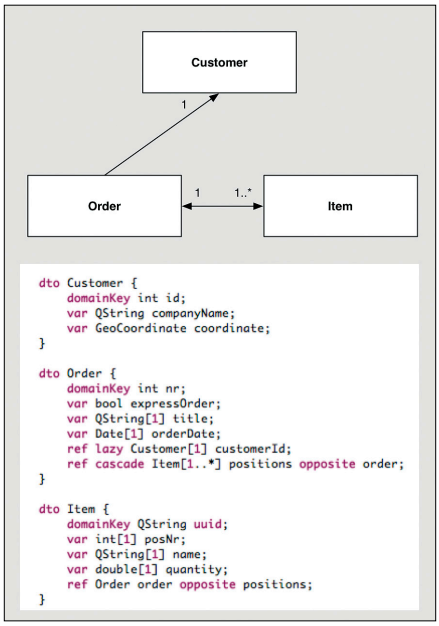
```
void MyObject::myFunction() {
  ...
  emit mySignal();
}
```

Slots, die Signale empfangen und verarbeiten können, werden wie ganz normale Methoden definiert, allerdings mit dem Keyword *slots* in der Header-Datei versehen:

```
class MyObject: public QObject {
  Q_OBJECT
  ...
public slots:
  void mySlotHandler();
  ...
}
```



Signale und Slots: Kopplung eines Sliders an einen Button (Bild 1)



Minimodell: Ausschnitt aus dem Datenmodell (Bild 2)

Dieses Konzept kann sowohl in C++ als auch in QML genutzt werden. In vielen Fällen verknüpft man Signale und Slots vollkommen unbemerkt. Soll beispielsweise ein Container nur sichtbar sein, wenn eine Checkbox angeklickt wurde, dann werden nur die entsprechenden Properties gesetzt:

```
CheckBox {
  id: calSyncCheckbox
  onCheckedChanged: {
    // do something
  }
} // end calSyncCheckbox

Container {
  id: calDefaultContainer
```

```
  visible: calSyncCheckbox.checked
  topPadding: 40
  bottomPadding: 20
  ...
}
```

An der CheckBox gibt es ein Signal *checked()*. Zu diesem Signal existiert direkt an der CheckBox ein Slot *onCheckedChanged()*, wodurch wir wiederum beliebige andere Funktionen aufrufen können.

Am Container gibt es die Property *visible*, deren Wert *true* oder *false* ist. Im Beispiel ist die Property *visible* direkt an das Signal *checked()* der CheckBox gekoppelt, und sobald der User die CheckBox anklickt, wird der Container ohne weiteren Programmieraufwand sichtbar. Natürlich hat auch die *visible*-Property des Containers ein *visibleChanged()*-Signal, und im Slot *onVisibleChanged()* könnten weitere Aktionen erfolgen.

Durch geschickte Kombinationen der bereits eingebauten Signale und Slots an den Standard-UI-Controls lassen sich bereits ohne großen Aufwand flexible Bedienoberflächen designen, die immer nur das anzeigen, was wichtig ist, oder automatisch die Farbe eines Fonts ändern, wenn an einem anderen Objekt ein Ereignis aufgetreten ist. Bild 1 zeigt die Kopplung eines Sliders an einen Button.

Das Arbeiten mit Signals und Slots ist sehr einfach und intuitiv und wird auch genutzt, um UI-Controls mit Datenobjekten zu verbinden.

Das Datenobjekt

Um es überschaubar zu gestalten, betrachten wir hier nur einen kleinen Ausschnitt aus dem Beispielprojekt. Die Gesamtübersicht finden Sie im Heft 11/2015 der **web & mobile developer** auf Seite 91.

Bild 2 zeigt unser Minimodell, das wir in diesem Teil verwenden und das sich nur auf Auftrag, Positionen und Kunde bezieht. Wenn Sie alles selbst nachvollziehen möchten, laden Sie sich das Beispielprojekt aus GitHub herunter und reduzieren das DSL-Datenmodell entsprechend. Das Bild enthält nicht nur das Klassendiagramm, sondern auch das Datenmodell für unsere DSL.

Fangen wir mit dem kleinsten Datenobjekt an – dem Kunden (Customer). Auch bei diesem recht einfachen Beispiel werden wir sehen, wie viel Arbeit uns die DSL abnimmt. Was wird alles generiert? In der Header-Datei stehen zunächst Angaben für das *Q_PROPERTY*-Makro:

```
Q_PROPERTY(int id READ id WRITE setId NOTIFY idChanged
FINAL)
Q_PROPERTY(QString companyName READ companyName WRITE
setCompanyName NOTIFY companyNameChanged FINAL)
Q_PROPERTY(GeoCoordinate* coordinate READ coordinate
WRITE setCoordinate NOTIFY coordinateChanged FINAL)
```

Die Anweisung *Q_PROPERTY* enthält als Erstes den Datentyp und den Property-Namen *int id*. Danach folgen die Methoden für *READ* und *WRITE* – also die Getter und Setter:

Datenobjekte in QML: Der Type ist bekannt, und damit auch alle Properties (Bild 3)

```
READ id WRITE setId
```

Als Letztes wird der Name des Signals angegeben, das automatisch bei Änderungen an der Property gefeuert wird:

```
NOTIFY idChanged
```

Die Variable, die Methoden und das Signal müssen aber noch definiert werden – hier für die Property *id*:

```
public:
  int id() const;
  void setId(int id);
```

Listing 1: Listen-Properties

```
import org.ekkescorner.data 1.0

Page {
  id: myOrderPage
  property Order myOrder
  property Item myItem

  Container {
    Label {
      text: "Positions: " +
      myOrder.positionsPropertyList.length
    }
  }

  function processPositions() {
    var summary = 0.0
    for (var i=0;
    i<myOrder.positionsPropertyList.length; i++) {
      myItem = myOrder.positionsPropertyList[i]
      summary += myItem.quantity
    }
  }
}
```

```
Q_SIGNALS:
  void idChanged(int id);

private:
  int mId;
```

Und die nachfolgenden Codezeilen sind die Implementierung in der *.cpp*:

```
Customer::Customer(QObject *parent) : QObject(parent),
mId(-1)
{

}
int Customer::id() const
{
  return mId;
}

void Customer::setId(int id)
{
  if (id != mId)
{
    mId = id;
    emit idChanged(id);
  }
}
```

Der Einsatz der DSL hat den Vorteil, dass wir für all dies nur eine Zeile Code benötigen:

Listing 2: Elemente der Liste

```
QDeclarativeListProperty<Item>
Order::positionsPropertyList()
{
  return QDeclarativeListProperty<Item>(this, 0,
  &Order::appendToPositionsProperty,
    &Order::positionsPropertyCount,
    &Order::atPositionsProperty,
    &Order::clearPositionsProperty);
}
void Order::appendToPositionsProperty
(QDeclarativeListProperty<Item> *positionsList,
Item* item)
{
  Order *orderObject = qobject_cast<Order *>
  (positionsList->object);
  if (orderObject) {
    item->setParent(orderObject);
    orderObject->mPositions.append(item);
    emit orderObject->addedToPositions(item);
  } else {
    qWarning() << "cannot append Item* to positions "
    << "Object is not of type Order*";
  }
}
int Order::positionsPropertyCount
(QDeclarativeListProperty<Item> *positionsList)
{
  Order *order = qobject_cast<Order *>
  (positionsList->object);
  if (order) {
    return order->mPositions.size();
  } else {
    qWarning() << "cannot get size positions "
    << "Object is not of type Order*";
  }
  return 0;
}
Item* Order::atPositionsProperty
(QDeclarativeListProperty<Item> *positionsList,
int pos)
{
  Order *order = qobject_cast<Order *>
  (positionsList->object);
  if (order) {
    if (order->mPositions.size() > pos) {
      return order->mPositions.at(pos);
    }
    qWarning() << "cannot get Item* at pos " << pos <<
    " size is "
      << order->mPositions.size();
  } else {
    qWarning() << "cannot get Item* at pos " << pos <<
    "Object is not of type Order*";
  }
  return 0;
}
void Order::clearPositionsProperty
(QDeclarativeListProperty<Item> *positionsList)
{
  Order *order = qobject_cast<Order *>
  (positionsList->object);
  if (order) {
    // positions are contained - so we must delete
    // them
    for (int i = 0; i < order->mPositions.size(); ++i)
    {
      order->mPositions.at(i)->deleteLater();
    }
    order->mPositions.clear();
  } else {
    qWarning() << "cannot clear positions " <<
    "Object is not of type Order*";
  }
}
```

```
dto Customer {
  domainKey int id;
...
}
```

Bei umfangreichen Datenobjekten kann das sonst ganz schön in Arbeit ausarten, und wenn dann Änderungen vorgenommen werden, vergisst man häufig das eine oder andere. Eine DSL hilft also auch dabei, schneller Refactorings vorzunehmen.

Wie bereits in den ersten Teilen beschrieben, generiert die DSL auch den Code, um die Datenobjekte dem UI (QML) bekanntzumachen:

```
qmlRegisterType<Customer>("org.ekkescorner.data", 1, 0,
"Customer");
```

Jetzt kann auf den Customer wie auf ein Javascript-Objekt zugegriffen werden – aber mit dem Vorteil, dass der Type bekannt ist und damit auch alle Properties – wie Bild 3 zeigt. Ebenso sind alle *events* (Signals) bekannt:

```
myCustomer.onCompanyNameChanged:
{
  console.debug("Company Name changed to " +
  myCustomer.companyName)
}
```

Etwas komplizierter wird der Einsatz von *Q_PROPERTIES*, wenn es sich um Collections (0..*) handelt. Ohne *Q_PROPERTIES* müssten Collections als Listen vom Typ *QVariantList* an das UI übergeben werden, und dort könnte darauf wie auf ein JavaScript-Array zugegriffen werden. Allerdings wäre der Type nicht bekannt und jedes Element wäre ein normales JavaScript-Object.

Listen-Properties

Um auch Listen typsicher übergeben zu können, gibt es ein ganz besonderes *Q_PROPERTY* vom Typ *QDeclarativeListProperty*. Schauen wir uns das beim Auftrag (Order) an, der eine Collection an Positionen (Items) enthält. Hier zunächst die *Q_PROPERTY* dazu:

```
Q_PROPERTY(QDeclarativeListProperty<Item>
positionsPropertyList READ positionsPropertyList
CONSTANT)
```

Wir definieren die *Q_PROPERTY* als eine Property vom Typ *QDeclarativeListProperty<Item>* und dazu einen Getter *(READ)*. Es gibt keine Signale und keine Setter *(WRITE)* – aber wozu haben wir eine DSL? Im UI greifen wir auf die Liste zu wie auf ein JavaScript-Array und erhalten jeweils Datenobjekte vom type *Item*. Listing 1 zeigt ein kleines Beispiel.

Sieht aus wie JavaScript, ist JavaScript – arbeitet aber unter der Haube mit kompilierten C++-Datenobjekten. Diese Datenobjekte werden übrigens als Pointer übergeben, können in mehreren UI-Controls im Einsatz sein, und bei ei- ▶

ner Änderung wirkt sich diese sofort überall aus – ob nun QML oder C++. Aber nochmal zurück zur Implementierung von *QDeclarativeListProperty*. Hier aus der Header-Datei:

```
public:
  QDeclarativeListProperty<Item>
  positionsPropertyList();
private:
  QList<Item*> mPositions;
```

So weit sieht es ja noch ganz normal aus – aber es werden ein paar zusätzliche statische Methoden benötigt:

```
static void appendToPositionsProperty
(QDeclarativeListProperty<Item> *positionsList, Item*
item);
static int positionsPropertyCount
(QDeclarativeListProperty<Item> *positionsList);
static Item* atPositionsProperty
(QDeclarativeListProperty<Item> *positionsList, int
pos);
static void clearPositionsProperty
(QDeclarativeListProperty<Item> *positionsList);
```

Sehen wir uns dazu die Implementierung in der *.cpp* an, dann wird schnell klar, warum diese Variante der *Q_PROPERTY* so selten im Einsatz ist (Listing 2).

All dieser Programmieraufwand ist notwendig, um auf die einzelnen Elemente der Liste zugreifen zu können oder die Liste zu löschen. Leider haben wir damit aber noch keinen schreibenden Zugriff auf die Elemente und bekommen auch keine Signale, wenn sich etwas ändert.

Also bauen wir das selbst, und da es dann einfach ist, dies als Pattern in die DSL einzufügen, wird es by magic bei Listen sofort mitgeneriert. Hier sind einige der zusätzlichen Methoden und Signale:

```
public:
  Q_INVOKABLE
  Item* createElementOfPositions();
  Q_INVOKABLE
  void undoCreateElementOfPositions(Item* item);
  Q_INVOKABLE
  void addToPositions(Item* item);
  Q_INVOKABLE
  bool removeFromPositions(Item* item);
  QList<Item*> positions();
  void setPositions(QList<Item*> positions);
Q_SIGNALS:
  void positionsChanged(QList<Item*> positions);
  void addedToPositions(Item* item);
  void removedFromPositionsByUuid(QString uuid);
```

Das Makro *Q_INVOKABLE* stellt sicher, dass diese Methode aus dem UI heraus aufgerufen werden kann. Listing 3 zeigt exemplarisch die Implementierung von *add/remove*, um Positionen hinzuzufügen oder zu entfernen und das UI darüber zu informieren. Jedes Mal, wenn ich mir den Code zur Verwaltung von Listen anschaue, weiß ich, warum ich die DSL entwickelt habe.

Listing 3: add/remove

```
void Order::addToPositions(Item* item)
{
  mPositions.append(item);
  emit addedToPositions(item);
}

bool Order::removeFromPositions(Item* item)
{
  bool ok = false;
  ok = mPositions.removeOne(item);
  if (!ok) {
   qDebug() << "Item* not found in positions";
   return false;
  }
  emit removedFromPositionsByUuid(item->uuid());
  // positions are contained - so we must delete
  // them
  item->deleteLater();
  item = 0;
  return true;
}
```

Fazit

Wieder haben wir gesehen, welche Vorteile eine DSL bietet und was für eine Leichtigkeit sich in einem Projekt ausbreitet, wenn man sich für alle Patterns nur einmal die Arbeit machen muss. ■

Links zum Thema

- DSL-Download
  http://github.com/lunifera/lunifera-dsl-extensions/tree/development
- DSL Sample Project
  http://github.com/ekke/ekkes_dsl_sample

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

DWX Developer Week 2016

20.-23. Juni 2016,
Messe Nürnberg

Call for Papers bis 10.01.2016

Jetzt Sessions einreichen!

- **Better Coding** (Prototyping, Code Reviews, Codequalität, Kodierrichtlinien)
- **Architektur** (Code strukturieren, Abhängigkeiten, skalieren, Konzepte, Methoden, MVC, Schichten, Hexagonal, SOA, CQRS, Event Sourcing, Microservices)
- **Best Practices** (RAD, Craftsmanship, Clean Code)
- **Sprachen** (C#, VB, F#, neue Sprachen)
- **Datenzugriff** (Entity Framework, ORM, Data Models)
- **Performance** (skalierbar, high performance, server - cloud, LAMP stack, Infrastrukturen)
- **Crossplattform** (web, hybrid, native, SOA, Apache Device Map, Titanium, PhoneGap, Sencha, QT etc.)
- **Mobile Architekturen** (mobile Middleware, Infrastruktur, Backend)
- **DevOps** (Virtualisierung, Docker, VMWare, etc.)
- **Testing** (TDD, ATDD, Unit-Test, Load-Test, Integrationstest, Systemtest)
- **Cloud/Server** (Cloud-Dienste, Datenbanken)

developer-week.de | **DeveloperWeek**

Aussteller & Sponsoren:

adorsys · brainLight. LIFE IN BALANCE · GFU Cyrus AG · MATHEMA · SPARX SYSTEMS

Veranstalter:

developer media
Neue Mediengesellschaft Ulm mbH

Präsentiert von:

dnc dotnetpro
web & mobile DEVELOPER

BENUTZERSCHNITTSTELLE LISTENANSICHT

Strukturhilfe

Wie Sie mit dem Listview-Control eine hilfreiche App erstellen.

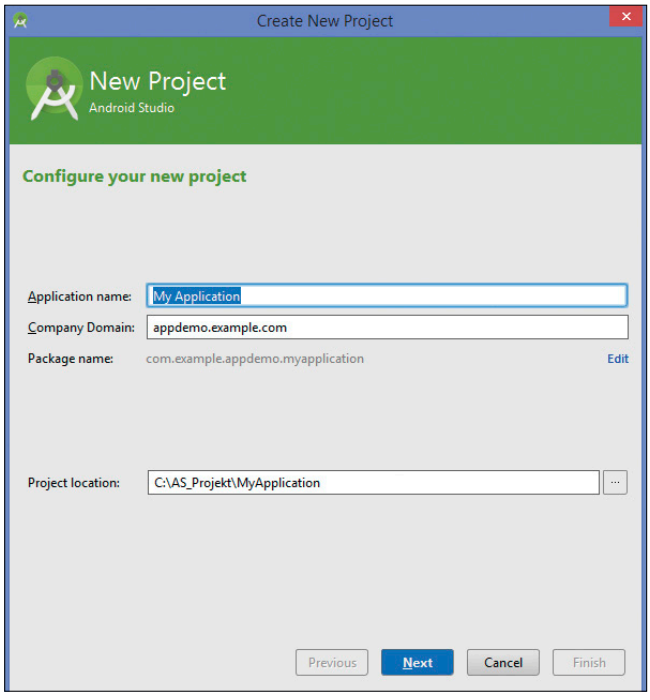
Es gibt etliche Ratgeber und Hinweise für die Optimierung von Arbeitsabläufen, Strukturen und To-do-Listen im privaten und beruflichen Umfeld. Gibt man unter Google den Suchbegriff *to-do-liste* oder *Strukturliste* ein, so erhält man Millionen von Suchmaschinenergebnissen. Was liegt also näher, als sich eine eigene Aufgabenliste für das Smartphone zu erstellen?

Grundfunktionalitäten

Bei der mobilen App-Entwicklung ist stets an die Zielgruppe zu denken. Aus der Zielgruppe heraus ergeben sich die benötigten Anforderungen an die App, deren Struktur und Inhaltstiefe.

Dieser Workshop kümmert sich nur um die technische Umsetzung einer App auf einem Android-Zielsystem. Der Inhalt beschränkt sich auf die Grundfunktionalitäten und Prozesse in der App.

Auch bei der Entwicklung von Apps arbeitet man am effizientesten, wenn man die Standardelemente der Benutzerschnittstelle des Android-Betriebssystems nutzt. Das bedeutet: In der Konzeptionsphase sollten die Standardelemente genutzt werden, die schon von Android vorgegeben werden. So sparen Sie viel unnötigen Aufwand während der Entwicklung.



Create New Project: Anlegen eines neuen Projekts (Bild 1)

Bedenken Sie, dass individuelle Interaktionselemente viel mehr Zeit bei der Entwicklung kosten.

Weiterhin sollten Sie auch als App-Entwickler darauf bedacht sein, die Businesslogik vom Frontend so weit wie möglich zu entkoppeln. Auch sollte die Usability der App nicht vernachlässigt werden. Man sollte bei jeder guten App mit nur wenigen Clicks zur gewünschten Funktion kommen. Das Ziel heißt bei der App-Entwicklung: isolieren, simplifizieren und optimieren.

Android Studio

Mit Hilfe der von Google genehmigten IDE (Integrated Development Environment) Android Studio von Jet Brains können auch Programmieranfänger sehr schnell zu guten, laufähigen Ergebnissen bei der App-Entwicklung kommen. Die Entwicklungsumgebung bietet einen schnellen und unkomplizierten Einstieg in die Programmierung. Die IDE ist mit folgenden Features ausgestattet:

- erweiterte Build-Tools,
- leistungsstarke Quellcode-Entwicklung (Smart-Editor, Code Refactoring, Code-Analyse, IntelliSense-Unterstützung),
- eine schnelle grafische Bedienoberfläche und ein Layout-Editor,
- einfacher Zugriff auf Google-Dienste.

Weiterhin zählen auch erweitertes Debugging und Import-Möglichkeiten dazu. Mit dem Layout-Editor von Android Studio lässt sich die grafische Oberfläche in Android deklarativ in XML definieren und so schnell und flexibel das gewünschte Layout anpassen.

Hierbei fühlt sich der Editor flüssig und komfortabel an und hilft somit auch bei der Zusammenstellung der Views und Konfiguration der Steuerelemente.

Die eigene App

In diesem Workshop wird davon ausgegangen, dass Android Studio zur Verfügung steht und installiert ist. Den Download und Hilfe zur Installation von Android Studio finden Sie unter http://developer.android.com/tools/studio.

Die Beispiel-App soll aufzeigen, wie schnell und effektiv, mit ein wenig Programmiererfahrung eine einfache App für Android erstellt werden kann.

Einfach heißt in diesem Fall aber nicht billig, sondern meint, wie für Apps immer wieder gefordert, ein gutes Gleichgewicht zwischen Funktionalität und Einfachheit.

Die App soll Ihnen die Erstellung und Verwaltung einer Strukturliste zur Verfügung stellen.

Hierfür benötigen Sie ein Eingabefeld, einen Bestätigungsschalter für die Eingabeübernahme und eine Liste zur Anzeige. Mit den Bedienelementen von Android lässt sich so etwas sehr einfach gestalten. Sie können die Oberfläche mit Hilfe des Layout-Editors erstellen oder die Objekteigenschaften der Controls auf Quelltextebene über XML modellieren.

Projekt anlegen

Unter einem Projekt werden immer alle Artefakte der Anwendung zusammengefasst. Dazu gehören die Quelltexte, Konfigurationsdateien und auch Grafiken, Sounds und Animationen.

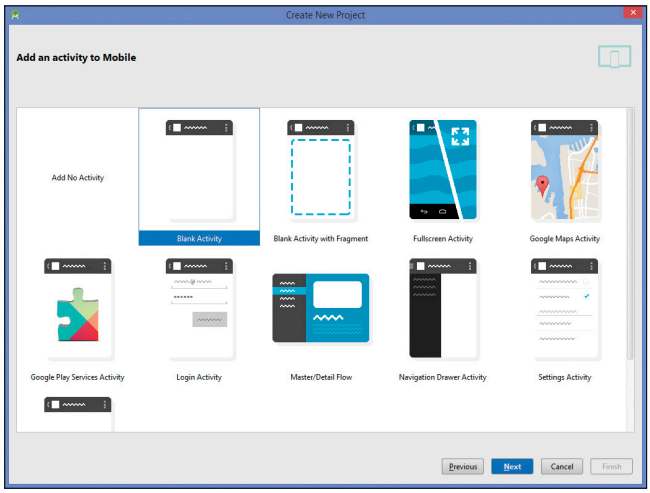
Legen Sie für die Beispiel-App ein neues Projekt an. Wählen Sie hierfür im Startbildschirm einfach *Start a new Android Studio project* oder über die Menüleiste *File* und *New Project*. Daraufhin öffnet sich der Dialog *Create New Project* wie in Bild 1. Hier legen Sie die grundlegenden Eigenschaften Ihres Projekts fest. Der Application-Name wird später auf dem Gerät angezeigt und bildet den Namen für die App. Wählen Sie hier einen prägnanten Namen für Ihre App. Im Beispiel verwenden wir den bescheidenen Begriff *Meine Liste*.

Die Klassen und Dateien werden unter Android, wie bekannt, in Paketen *(Packages)* abgelegt. Seien Sie daher bei der Vergabe des Package-Namens besonders sorgfältig, vor allem, wenn Sie die App im Google Play Store veröffentlichen möchten. Der Paketname muss eindeutig sein.

Für das Beispiel behalten Sie den vorgeschlagenen Package-Namen bei. Unter *Project location* legen Sie einfach den Speicherort des Projekts fest.

Über *Next* rufen Sie die zweite Seite des Projekt-Assistenten auf.

Dort legen Sie die Gerätekategorien fest, für die Ihre App zur Verfügung stehen soll. Wählen Sie für das Beispiel die Kategorie *Phone and Tablet* aus und legen Sie das Minimum SDK fest. Die App verwendet das API 15 und unterstützt somit 87,9 Prozent aller Android-Devices. Plattformen können mit dem Android SDK-Manager installiert und gelöscht werden. Sie sollten für das Beispiel keine SDK-Version unterhalb von Android 4 verwenden.



Anlegen der gewünschten Activity (Bild 2)

Listing 1: strings.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
  <string name="app_name">Meine Liste</string>
  <string name="action_delete_done_tasks">
  Aufgabe löschen</string>
  <string name="action_delete_all">
  Alle Einträge löschen</string>
  <string name="context_delete">Löschen</string>
  <string name="context_edit">Bearbeiten</string>
  <string name="add_task">Aufgabe hinzufügen
  </string>
</resources>
```

Ein Klick auf *Next* ruft den Assistenten *Add an activity to Mobile* auf (Bild 2). Markieren Sie für das Beispiel *Blank Activity* und klicken Sie auf *Next*.

Im Dialogfenster *Choose options for your new file* konfigurieren Sie dann die ausgewählte Activity. Vergeben Sie hier den Klassennamen der Activity, den Namen der Layoutdatei, sowie einen Titel und den Namen für die Menüeinträge-Datei. Die Layout- und die Menü-Ressourcen-Datei werden im XML-Format angelegt. Das Beispiel übernimmt die vorgeschlagenen Einstellungen. Mit *Finish* schließen Sie die Projektanlage ab.

Projektstruktur

Android-Apps werden immer zu einer baumartigen Struktur zusammengefasst. Hierbei bietet das Werkzeugfenster mehrere Sichten an. Die Ansicht *Project* entspricht dem Aufbau der Ebenen innerhalb des Dateisystems, während die Sicht *Android* die Struktur eines Projekts abbildet. So erhalten Sie einen schnellen Zugriff auf die wichtigsten Dateien und Verzeichnisse.

Bei Apps stellt die Bedienoberfläche immer das Aushängeschild einer Anwendung dar. Hierzu zählen vor allem Bilder, Symbole, Texte und Steuerelemente. Sie sollten immer so eingesetzt werden, dass sie den Anwender bei der Bedienung der App anleiten und unterstützen.

Unter Android sollte Text immer zentral in der Datei *strings.xml* abgelegt werden. Sie befindet sich im Verzeichnis *res/values*. Passen Sie die Datei für die App wie in Listing 1 dargestellt an. Das Attribut *name* des Elements *<string>* wird im Quelltext als Bezeichner verwendet.

Layout

Die Layoutbeschreibung kann auch vollständig in der Sprache XML modelliert werden. Die entsprechenden Dateien, die mit *layout* beginnen, werden im Unterverzeichnis von *res* abgelegt.

Das folgende Beispiel *activity_main.xml* ist Bestandteil des Projekts und soll für die App wie in Listing 2 aufgeführt abgeändert werden. Sie stellt das allgemeine Design der Activity dar. ▶

Hierbei ist die Grundstruktur absichtlich sehr übersichtlich gehalten worden. Weiterhin benötigt die App noch ein List- und Edit-Layout. Diese beiden Dateien können Sie über das Kontextmenü *New, Menu resource file* im Layoutverzeichnis *res/layout* anlegen. Listing 3 zeigt den Aufbau als XML für die Struktur der Liste und Listing 4 das Layout für das Editieren der Liste. Wird, wie im Beispiel, die Bedienoberfläche in einer XML-Datei deklariert und erst zur Laufzeit zu einem Objektbaum entfaltet, kann man die Referenzen auf die Komponenten über die Methode *findViewById()* ermitteln. Der ihr übergebende Wert entspricht in der Regel einer Konstanten aus *R.id*.

Listing 2: Das XML-Layout der App

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.
 com/apk/res/android"
 android:orientation="vertical"
 android:layout_width="fill_parent"
 android:layout_height="fill_parent"
>

  <LinearLayout
   android:orientation="horizontal"
   android:layout_width="fill_parent"
   android:layout_height="wrap_content"
   android:gravity="center_horizontal"
  >

    <EditText
     android:id="@+id/input_task"
     android:layout_width="0dp"
     android:layout_height="wrap_content"
     android:layout_weight="0.8"
     android:hint="@string/add_task"
    />

    <ImageButton
     android:id="@+id/add_task_button"
     android:layout_width="0dp"
     android:layout_height="wrap_content"
     android:layout_weight="0.2"
     android:src="@drawable/ic_action_accept"
    />

  </LinearLayout>
  <ListView
   android:id="@+id/list_view"
   android:layout_width="fill_parent"
   android:layout_height="wrap_content"
   android:clickable="true"
  />

</LinearLayout>
```

Das heißt also, die Basisklasse aller Bedienelemente ist *android.view.View*. Somit besteht die Bedienoberfläche einer App aus einer oder mehreren Views oder von ihr abgeleiteten Klassen.

Programmlogik

Unter Android besteht eine App aus mindestens einer Activity. Je nach Funktionsumfang kann auch eine Vielzahl von Activities in einer App enthalten sein. Hierbei ist die Activity einer Bedienoberfläche aus Views oder ViewGroups zugeordnet. Somit stellen Activities die Grundbausteine einer App dar, die sich gegenseitig aufrufen können.

Da im Beispiel nur eine Activity benutzt wird, können wir jetzt beginnen, die Klasse *MainActivity* für die App zu Implementieren. Übernehmen Sie hierzu die Klasse *MainActivity* aus Listing 5.

Als Erstes werden in der Klasse die benötigten Variablen der App deklariert. Die Initialisierung wird in der *onCreate()*-Methode der Activity vorgenommen. Die fehlende Implementierung des ListAdapters folgt später.

Weiterhin müssen Sie auf das Anklicken der Schaltfläche reagieren. Hierfür wird ein OnClickListener registriert. Dieses Interface besteht aus der Methode *onClick()* und reagiert auf die Benutzeraktion.

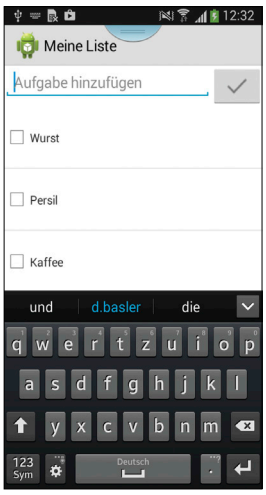
Dann benötigen wir in der App noch ein Optionsmenü und ein Kontextmenü für die Bearbeitung der Listeneinträge. Wenn eine Activity ein Optionsmenü anbieten möchte, müs-

Listing 3: Layout für den Inhalt der Liste

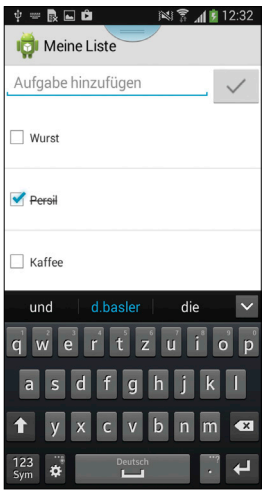
```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.
 com/apk/res/android"
 android:orientation="horizontal"
 android:layout_width="fill_parent"
 android:layout_height="fill_parent"
 android:paddingTop="20dp"
 android:paddingBottom="20dp"
>

  <CheckBox
   android:id="@+id/list_checkbox"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:focusable="false"
  />

  <TextView
   android:id="@+id/list_task_view"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:focusable="false"
  />

</LinearLayout>
```

sen Sie in der Activity die Methode *onCreateOptionsMenu()* überschreiben. Hier wird das Menü entfaltet, dessen Elemente in der Datei *actions.xml* definiert wurden. Sie wird unter *res/menu* abgelegt und hat folgenden Aufbau:

```
<menu xmlns:android="http://
schemas.android.com/apk/res/
android">

  <item android:id="@+id/action
   _delete_done_tasks"
   android:title="@string/action
   _delete_done_tasks"
   android:showAsAction="never"
  />

  <item android:id="@+id/action_delete_all"
   android:title="@string/action_delete_all"
   android:showAsAction="never"
  />

</menu>
```



Erstellen einer Aufgabenliste (Bild 3)



Erledigt: Eine Aufgabe als erledigt markieren (Bild 4)

Android unterstützt auch das Konzept von Kontextmenüs. Hier wird über das lange Antippen, also Tippen und Halten, eines Elements das Menü geöffnet.

Besonders gerne werden ListViews mit Kontextmenüs versehen, so, wie es auch in der Beispiel-App vorgesehen ist. Der Bau des Kontextmenüs verläuft hierbei analog zum Optionsmenü. Sie brauchen bloß die Methoden *onCreateContextMenu()* und *onContextItemSelected()* zu überschreiben beziehungsweise zu implementieren. Die *context_menu.xml* Datei enthält die Elemente für das Kontextmenü und wird unter *res/menu* abgelegt.

Das folgende Listing zeigt die benötigten Elemente:

```
<?xml version="1.0" encoding="UTF-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/
android"
>
  <item android:id="@+id/context_delete"
  android:title="@string/context_delete"
  />
  <item android:id="@+id/context_edit"
  android:title="@string/context_edit"
  />
</menu>
```

Somit ist das Grundgerüst für die App erstellt. Jetzt können Sie die Aufgaben-Klasse für die Strukturliste erzeugen. Diese Klasse sorgt für das Speichern der eingetragenen Textinhalte. Legen Sie hierfür, über das Kontextmenü *New, Java Class* im Source-Verzeichnis eine neue Java-Klasse mit dem Namen *Aufgabe* an. Das folgende Listing zeigt die benötigte Implementierung:

```
public class Aufgabe {
  private String aufgabeContent;
  private boolean isDone;
  public Aufgabe(String aufgabeContent, boolean isDone) {
    this.taskContent = aufgabeContent;
    this.isDone = isDone;
  }
  public String getTaskContent() {
    return aufgabeContent;
  }
  public boolean isDone() {
    return isDone;
  }
```

▶

Listing 4 für das Editieren der Liste

```
<?xml version="1.0" encoding="UTF-8"?>
<LinearLayout xmlns:android=
 "http://schemas.android.com/apk/res/android"
 android:orientation="vertical"
 android:layout_width="fill_parent"
 android:layout_height="fill_parent"
 android:id="@+id/edit_dialog_layout"
 android:paddingLeft="10dp"
 android:paddingTop="20dp"
 android:paddingBottom="20dp"
 android:paddingRight="10dp"
>

  <TextView
   android:id="@+id/edit_message"
   android:layout_width="fill_parent"
   android:layout_height="wrap_content"
   android:textSize="20sp"
   android:layout_marginBottom="20dp"
   android:text="Edit Task"
  />

  <EditText
   android:id="@+id/edit_input"
   android:layout_width="fill_parent"
   android:layout_height="wrap_content"
   android:singleLine="false" >
     <requestFocus />
  </EditText>

</LinearLayout>
```

```
  public void setTaskContent(String aufgabeContent) {
    this.aufgabeContent = aufgabeContent;
  }
  public void setIsDone(boolean isDone) {
    this.isDone = isDone;
  }
}
```

Hierbei übernimmt der String *aufgabeContent* das Speichern des Textinhalts. Über *isDone* wird angegeben, ob der Eintrag in der Liste erledigt ist, und die *Getter*- und *Setter*-Methoden werden ebenfalls implementiert.

Listing 5: Die Klasse MainActivity der App (Teil 1)

```
import java.util.ArrayList;
import java.util.List;
import android.app.ActionBar;
import android.app.Activity;
import android.app.AlertDialog;
import android.content.DialogInterface;
import android.os.Bundle;
import android.view.ContextMenu;
import android.view.LayoutInflater;
import android.view.Menu;
import android.view.MenuInflater;
import android.view.MenuItem;
import android.view.View;
import android.widget.AdapterView;
import android.widget.EditText;
import android.widget.ImageButton;
import android.widget.ListView;
import android.widget.TextView;

public class MainActivity extends Activity {
  ImageButton button;
  EditText input;
  ListView task_list_view;
  List<Aufgabe> tasks;
  AufgabeListAdapter adapter;
  ActionBar actionBar;

  @Override
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
    tasks = new ArrayList<Aufgabe>();
    button = (ImageButton) findViewById
    (R.id.add_task_button);
    input = (EditText) findViewById(R.id.input_task);
    task_list_view = (ListView) findViewById
    (R.id.list_view);
    registerForContextMenu(task_list_view);
    adapter = new AufgabeListAdapter(tasks, this);
    task_list_view.setAdapter(adapter);
```

Android stellt mit der *ListView* eine Komponente zur Verfügung, die auch mehrzellige Elemente und Grafiken darstellen kann. Auch eine *Checkbox* zum Anzeigen bestimmter Ereignisse ist möglich. In der Beispiel-App wird über eine *Checkbox* der Listeintrag auf erledigt gesetzt und der Textinhalt durchgestrichen.

ListView

Die Klasse *ListActivity* realisiert eine Listenansicht, die normalerweise den gesamten Bildschirm ausfüllt. Die Methode *getListView()* dieser Activity liefert eine Referenz auf ein zentrales Objekt des Typs *ListView*.

Um auf das Antippen eines Eintrags zu reagieren, wird ein *OnItemClickListener* registriert. Welche Daten eine Liste anzeigt, wird über einen Adapter gesteuert. Hierfür steht schon eine Vielzahl von fertigen Klassen zur Verfügung. Der Zugriff

Listing 5: Die Klasse MainActivity der App (Teil 2)

```
    button.setOnClickListener(new
    View.OnClickListener() {
      public void onClick(View v) {

        if (input.getText().length() > 0) {
          tasks.add(new Aufgabe
          (input.getText().toString(), false));
          adapter.notifyDataSetChanged();
          input.setText("");
        }

      }
    });
  }
  @Override
  protected void onPause() {
    super.onPause();
  }
  @Override
  public boolean onCreateOptionsMenu(Menu menu) {
    MenuInflater inflater = getMenuInflater();
    inflater.inflate(R.menu.actions, menu);
    return super.onCreateOptionsMenu(menu);
  }
  @Override
  public boolean onOptionsItemSelected(MenuItem
  item)
  {
    switch (item.getItemId()) {
      case R.id.action_delete_done_tasks:
      deleteDoneTasks();
      this.adapter.notifyDataSetChanged();
      break;
      case R.id.action_delete_all:
      this.tasks.clear();
      this.adapter.notifyDataSetChanged();
```

erfolgt über die Methode des Interfaces *android.widget.ListAdapter* auf einen Adapter und die durch ihn bereitgestellten Daten.

Jetzt benötigen Sie nur noch eine Klasse, die aus den Aufgaben die Einträge für die Liste vornimmt, die in der ListView angezeigt werden sollen.

Java-Klasse anlegen

Die Liste soll hierbei ganz einfach in eine ArrayList gespeichert werden. Legen Sie hierfür eine weitere Java-Klasse mit dem Namen *AufgabeListAdapter* an. Listing 6 zeigt, wie die Implementierung aussehen sollte. Die Klasse erbt von *BaseAdapter*. Im Konstruktor werden die Aufgaben in eine ArrayList abgelegt. Die Methode *getCount()* liefert die Länge der Liste und *getItem()* liefert das Element an einer bestimmten Position.

Die Methode *getView()* baut aus den Daten des Modells einen Eintrag zusammen und stellt ihn anschließend der View-Instanz zur Verfügung. Mit Hilfe eines LayoutInflators aus der XML-Datei *(list_layout.xml)* wird ein entsprechender Komponentenbaum erzeugt und der Variablen *convertView* zugewiesen.

Über die Methode *setTag(holder)* wird ein ViewHolder übergeben. Er fungiert als Platzhalter, um später einfach und effizient an die Elemente des Komponentenbaums zu gelangen. Somit ist unsere App fertig.

Bild 3 zeigt die fertige App auf einem Samsung Smartphone als Einkaufsliste. Sie können somit die Strukturliste ganz individuell einsetzen. Bild 4 zeigt das Abarbeiten der Liste und das Abhaken der Einträge. ▶

Listing 5: Die Klasse MainActivity der App (Teil 3)

```
      break;
    }
    return super.onOptionsItemSelected(item);
  }
  public void deleteDoneTasks() {
    for (int i = 0; i < tasks.size(); i++) {
      if (tasks.get(i).isDone()) {
        tasks.remove(i);
      }
    }
  }

  @Override
  public void onCreateContextMenu(ContextMenu menu,
  View v, ContextMenu.ContextMenuInfo menuInfo) {
    super.onCreateContextMenu(menu, v, menuInfo);
    MenuInflater inflater = getMenuInflater();
    inflater.inflate(R.menu.context_menu, menu);
  }
  @Override
  public boolean onContextItemSelected(MenuItem
  item) {
    AdapterView.AdapterContextMenuInfo info =
    (AdapterView.AdapterContextMenuInfo)
    item.getMenuInfo();
    int position = (int) info.id;
    switch (item.getItemId()) {
      case R.id.context_delete:
      this.tasks.remove(position);
      this.adapter.notifyDataSetChanged();
      break;
      case R.id.context_edit:
      createEditDialog(tasks.get(position));
      break;
    }
    return super.onContextItemSelected(item);
  }
  public void createEditDialog(final Aufgabe task) {
    LayoutInflater li =
    LayoutInflater.from(MainActivity.this);
```

Listing 5: Die Klasse MainActivity der App (Teil 4)

```
    View dialogView =
    li.inflate(R.layout.edit_layout, null);

    AlertDialog.Builder alertDialogBuilder = new
    AlertDialog.Builder(MainActivity.this);
    alertDialogBuilder.setView(dialogView);
    final EditText inputText = (EditText)
    dialogView.findViewById(R.id.edit_input);
    inputText.setText(task.getTaskContent());
    final TextView dialogMessage = (TextView)
    dialogView.findViewById(R.id.edit_message);
    alertDialogBuilder
    .setCancelable(true)
    .setPositiveButton("Speichern",
    new DialogInterface.OnClickListener() {
      public void onClick(DialogInterface dialog,
      int id) {
        task.setTaskContent
        (inputText.getText().toString());
        adapter.notifyDataSetChanged();
      }
    })
    .setNegativeButton("Abbrechen",
    new DialogInterface.OnClickListener() {
      public void onClick(DialogInterface dialog,
      int id) {
        dialog.cancel();
      }
    });
    final AlertDialog alert =
    alertDialogBuilder.create();
    alert.show();
  }
}
```

Listing 6: Die Klasse AufgabeListAdapter (Teil 1)

```
import android.content.Context;
import android.graphics.Paint;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.BaseAdapter;
import android.widget.CheckBox;
import android.widget.CompoundButton;
import android.widget.TextView;

import java.util.List;

public class AufgabeListAdapter extends BaseAdapter
{
  private final List<Aufgabe> tasks;
  private final LayoutInflater inflator;
  private Aufgabe task;

  public AufgabeListAdapter(List<Aufgabe> tasks,
  Context context) {
    this.tasks = tasks;
    inflator = LayoutInflater.from(context);
  }

  public int getCount() {
    return tasks.size();
  }

  public Object getItem(int position) {
    return tasks.get(position);
  }

  public long getItemId(int position) {
    return position;
  }

  public View getView(final int position, View
  convertView, ViewGroup parent) {
    final AufgabeListAdapter.ViewHolder holder;

    if (convertView == null) {
      convertView = inflator.inflate
      (R.layout.list_layout, parent, false);
      holder = new AufgabeListAdapter.ViewHolder();
      holder.task_view = (TextView)
      convertView.findViewById(R.id.list_task_view);
      holder.done_box = (CheckBox)
      convertView.findViewById(R.id.list_checkbox);
      convertView.setTag(holder);

    } else {
      holder = (AufgabeListAdapter.ViewHolder)
      convertView.getTag();
    }
```

Listing 6: Die Klasse AufgabeListAdapter (Teil 2)

```
    task = (Aufgabe) getItem(position);
    holder.done_box.setOnCheckedChangeListener(new
    CompoundButton.OnCheckedChangeListener() {
      public void onCheckedChanged(CompoundButton
      buttonView, boolean isChecked) {
      tasks.get(position).setIsDone(isChecked);

        if (isChecked) {
          holder.task_view.setPaintFlags
          (holder.task_view.getPaintFlags() |
          Paint.STRIKE_THRU_TEXT_FLAG);
        } else {
          holder.task_view.setPaintFlags(0);
        }
      }
    });

    holder.task_view.setText(task.getTaskContent());
    holder.done_box.setChecked(task.isDone());

    return convertView;
  }
  static class ViewHolder {

    TextView task_view;
    CheckBox done_box;
  }
}
```

Fazit

Mit Android Studio macht das Entwickeln von App einfach Spaß. Die speziell für Android ausgelegte IDE unterstützt die tägliche Arbeit und die Entwicklung von Apps durch ihre intuitive Bedienung sehr gut. Alle Funktionen sind klar gegliedert und ohne großen Einarbeitungsaufwand nutzbar. Auch der Layout-Editor und das IntelliSense sind ein großer Gewinn. Die Beispiel-App lässt sich ganz individuell erweitern und verbessern. Nutzen Sie neue und eigene Komponenten, speichern Sie die Werte in einer Datenbank oder als Datei.

Die gezeigten Implementierungen sind Standardimplementierungen der Methoden und lassen sich deshalb ohne Weiteres auf andere Apps oder Activities übertragen. ■

Daniel Basler
ist Senior Consultant für Microsoft-Technologien und beschäftigt sich darüber hinaus mit Datenbanken und Compiler-Bau.

Jetzt Aussteller werden!

developer-week.de | **DeveloperWeek**

Aussteller & Sponsoren 2015/2016:

adorsys · AIT · best-blu · brainLight · cellent · CENTIGRADE · centron · codecentric · DataAccess · DATEV · DevExpress · ECS Webhosting · embarcadero · ERGOSIGN · evosoft · GfK · GFU Cyrus AG · HANSER · HeiReS · ILNUMERICS · InnoGames · intel Software · INTERNETONE · it-agile · ITECH PROGRESS · LOVOO · MATHEMA · Microsoft · MINERVA · mitp · reddo · redgate · SafeNet · sevenval · SIGS DATACOM · SOS · SPARX SYSTEMS · THEOBALD SOFTWARE · UNIRENT

Veranstalter: developer media – Neue Mediengesellschaft Ulm mbH

Präsentiert von: dotnetpro · web & mobile DEVELOPER

REQUEST-BIBLIOTHEK HTTPFUL

Leichter ins Gespräch kommen

Immer mehr Systeme müssen miteinander per HTTP kommunizieren.

Bild: shutterstock@Dooder

Der Standard für die Kommunikation von Systemen über HTTP ist die Bibliothek libcurl. Diese ist recht mächtig, aber nicht leicht zu beherrschen. Darum gibt es für PHP-Programmierer eine Zusatzbibliothek, die eine saubere Abstraktion dazu zur Verfügung stellt.

Die PHP-Version der Bibliothek libcurl ist für die meisten Programmierer ausreichend, wenn sie in ihren Programmen einen Request zu einem anderen Server ausführen müssen. Doch jenseits der einfachen *GET*-Abfragen lauern schnell die Untiefen von libcurl. Das richtige Zusammenspiel der vielen Optionen, mit denen man per *curl_setopt()* das Verhalten steuern kann, ist kein Vergnügen, sofern man zum Beispiel eine REST-Schnittstelle ansprechen oder andere Besonderheiten nutzen möchte.

Mit httpful kommt da Erlösung in Sicht: Diese klassenorientierte Bibliothek vereinfacht die Arbeit deutlich, und der dabei verwendete Code ist auch noch viel besser verständlicher als der bei libcurl.

httpful in ein Projekt integrieren

Möchten Sie nur schnell in httpful einsteigen, ist der einfachste Weg die Nutzung des PHAR-Archivs, das Sie herunterladen und in Ihre Programme einbinden:

```
include 'httpful.phar';
```

Bei Verwendung von Composer fügen Sie entweder in der *composer.json* folgende Zeile zum Abschnitt *require* hinzu:

```
"nategood/httpful": "*"
```

Oder Sie laden httpful per Kommandozeile, etwa mittels:

```
composer require nategood/httpful
```

Wenn dann noch der mit Composer kommende Autoloader eingebunden ist, können Sie die Klassen von httpful direkt nutzen. Sollten Sie von Composer die Fehlermeldung erhalten, dass die libcurl-Library für PHP fehlt, dann müssen Sie das entsprechende Modul nachinstallieren. Auf Ubuntu beziehungsweise Debian zum Beispiel per:

```
sudo apt-get install php5-curl
```

Zunächst soll einfach nur die Startseite von Spiegel Online abgerufen werden. Der Code dazu sieht so aus:

```
include "vendor/autoload.php";
$url= "http://www.spiegel.de/";
$response = \httpful\Request::get($url) ->send();
```

Das zurückgelieferte Objekt ist vom Typ *httpful\Response* und ist schön aufgeräumt: Die Property *body* bietet einfach den zurückgelieferten Inhalt, hier eben den HTML-Quelltext der Spiegel-Seite. In der Eigenschaft *code* finden Sie den HTTP-Antwortcode des Servers.

Die Eigenschaft *headers* beinhaltet ein Array der verschiedenen, vom Server gesetzten Header. Um beispielsweise die Cache-Vorgaben herauszufinden, schreiben Sie

```
echo "Typ: " . $response
->headers['cache-control'];
```

Auch den Typ und den Zeichensatz des *body* finden Sie im Array *headers* über den bekannten Schlüssel *content-type*. Das Antwort-Objekt liefert allerdings beide Werte schön separiert direkt als einzelne Properties:

```
echo "Typ " . $response->content_type;
echo "Zeichensatz " . $response->charset;
```

Möchten Sie vor der weiteren Verarbeitung zunächst feststellen, ob die Server-Antwort einen Fehlerzustand mitteilt, nut-

zen Sie die Helper-Funktion *hasError()*. Um zu testen, ob der Body mit Leben gefüllt ist, befragen Sie stattdessen *hasBody()*. Schließlich liefert das Antwort-Objekt auch noch sein Gegenstück, also das zum Abruf verwendete Request-Objekt sowie einige Metadaten, wie etwa IP-Adresse und Port des Servers oder gestoppte Messwerte für das Timing der HTTP-Abfrage.

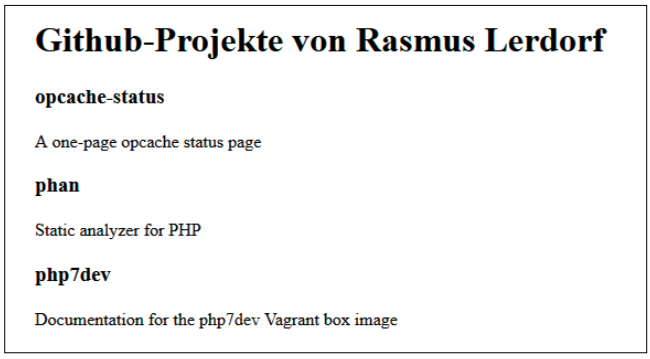
Mit einem API sprechen

Das erste Beispiel war für httpful natürlich nur ein Klacks und hätte im Prinzip genauso mit den Bordmitteln von PHP *(file_get_contents()* mit dem URL als Parameter) erledigt werden können. Als Nächstes soll deshalb ein richtiges API angesprochen werden. GitHub besitzt eine REST-Schnittstelle, die man ohne Anmeldung ansprechen darf. Dieses API soll Ihnen etwas zu Rasmus Lerdorf, dem Vater von PHP, erzählen.

```
$url = "https://api.github.com";
$url .= /users/rlerdorf/repos";
$response = \httpful\Request::get($url)
  ->expectsJson()
  ->send();
echo "<h1>Github-Projekte von Rasmus Lerdorf</h1>";
// Antwort auswerten
foreach($response->body as $repo){
  if (!$repo->fork){
    echo "<h3>{$repo->name}</h3> {$repo
    ->description}";
  }
}
```

Der wichtigste Unterschied zu vorhin liegt in der Gestaltung der Anfrage. Hier wird dem Request-Objekt mittels eines eingefügten *expectsJson()* mitgeteilt, dass eine Antwort im JSON-Format erwartet wird.

Die Wirkung dieser Einstellung besteht darin, dass die Bibliothek nach dem Erhalt der Antwort ein Parsing durchführt. Als Folge ist *body* nicht einfach ein String wie zuvor, sondern eine Datenstruktur in Objektform.



REST/JSON: Das zweite Beispiel fragt per REST/JSON Repositories eines GitHub-Autors ab (Bild 1)

Constants summary			
string	JSON	'application/json'	#
string	XML	'application/xml'	#
string	XHTML	'application/html+xml'	#
string	FORM	'application/x-www-form-urlencoded'	#
string	UPLOAD	'multipart/form-data'	#
string	PLAIN	'text/plain'	#
string	JS	'text/javascript'	#
string	HTML	'text/html'	#
string	YAML	'application/x-yaml'	#
string	CSV	'text/csv'	#

MIME-Typen: Mit diesen MIME-Typen kann httpful umgehen und wandelt die empfangenen Daten selbstständig in passende Datenstrukturen um (Bild 2)

In diesem Fall liefert der Aufruf ein Array aller Repositories des angesprochenen GIT-Eigentümers zurück, in der jedes Element Properties des jeweiligen Repositorys enthält, wie Namen, Beschreibung, URL et cetera.

Wie die Struktur des Bodys aussieht, hängt also immer vom API und dem jeweiligen Aufruf ab. Sie können den Aufbau entweder der API-Dokumentation entnehmen oder machen einfach testhalber eine Abfrage und lassen sich das Ergebnis dann mittels folgender Zeile anzeigen:

```
var_dump($response->body)
```

Weil im Beispiel nur die eigenen Projekte des Autors angezeigt werden sollen, nicht aber diejenigen, die nur ein Fork eines anderen Repositorys sind, wurde eine entsprechende Abfrage auf *$repo->fork* eingebaut (Bild 1).

Auf Content-Typen reagieren

Der im vorigen Skript per *expectsJson()* eingefügte Hinweis auf den Typ der zurückgelieferten Daten hat natürlich noch einige Kollegen. So können Sie unter anderem auch XML, YAML und CSV-Daten nutzen und schreiben dann entsprechend *expectsXml()* oder *expectsYaml()* (Bild 2).

Sie könnten den Fingerzeig auf den Inhaltstyp auch ganz weglassen. Dann orientiert sich httpful einfach am Header-Feld *content-type* in der HTTP-Antwort. Damit wäre Ihr Codeschnipsel flexibler, weil er selbstständig auf die gelieferten Daten reagiert.

Die Variante, den Typ festzuklopfen, ist dann notwendig, wenn der Inhaltstyp nicht korrekt geliefert wird oder man lieber einen Fehler erhalten möchte, wenn der erwartete Typ nicht kommt. Würden Sie zum Beispiel den Typ per *expectsXml()* festlegen, aber JSON-Daten bekommen, dann kracht es im verwendeten XML-Parser und er wirft eine Exception mit einem Fehlerhinweis wie *Unable to parse response as XML*.

Bislang waren alle Beispiele reine Leseabfragen mittels der HTTP-Methode *GET*. Die Bibliothek httpful unterstützt aber selbstverständlich auch die anderen Methoden *POST*, ▶

PUT, DELETE, HEAD und *OPTIONS*. Zur Auswahl der gewünschten Variante schreiben Sie diese einfach statt *::get()* an die Stelle, an welcher der Request erzeugt wird. Um beispielsweise eine Ressource zu löschen, verwenden Sie den folgenden Befehl:

```
$response = \httpful\Request::delete($url)
```

Die schreibenden HTTP-Methoden wie *POST* (Ressourcen anlegen) oder *PUT* (Ressourcen aktualisieren) erfordern, dass Sie Daten mitsenden. Dazu bietet die Library die Methode *body()*, die Sie in die Aufrufkette miteinflechten.

Nutzdaten mitsenden

Angenommen, Sie senden an ein API ein Update eines Artikels, der einen neuen Preis und Lagerbestand erhalten soll. Das API spricht XML. Dann kann der zugehörige Aufruf so aussehen:

```
$url = "http://myapi.xy/artikel/123";
$daten="<xml>
  <price>29.99</price>
  <amount>999</amount>
</xml>";
$response = \httpful\Request::put($url)
  ->body($daten)
  ->sendsXml()
  ->send();
```

Die Methode *sendsXml()* sorgt einerseits dafür, dass der Inhaltstyp für die Anfrage korrekt gesetzt wird, und führt bei Bedarf eine automatische Umwandlung der Daten durch: Wenn Sie dem Aufruf von *body()* keinen XML-String, sondern ein assoziatives Array als Parameter mitgeben, dann wandelt httpful die Daten automatisch in XML um. Das klappt genauso mit JSON und im Prinzip auch mit CSV.

Bei Verwendung von *sendsCsv()* müssen die Daten als zweidimensionales Array geliefert werden. Dabei drückt die eine Dimension die Zeilen aus und die andere die Werte innerhalb der Zeilen. Wenn Sie also etwa einem API die Lieferung von Büroartikeln mitteilen möchten, könnte das folgendermaßen aussehen:

```
$url = "http://myapi.xy/lieferung";

$daten=[
  ['artikelnr'=>'9981','menge'=>25,'name'=>
  'Radiergummi'],
  ['artikelnr'=>'2312','menge'=>100,'name'=>
  'Mine 0,7 mm'],
  ['artikelnr'=>'0081','menge'=>10,'name'=>
  'Druckerpapier A4']
];

$response = \httpful\Request::post($url)
  ->body($daten)
  ->sendsCsv()
  ->send();
```

Die Library macht dann daraus in der Anfrage den folgenden Inhalt:

```
artikelnr,menge,name
9981,25,Radiergummi
2312,100,"Mine 0,7 mm"
0081,10,"Druckerpapier A4"
```

Sie sehen, dass die Headerzeile automatisch aus den Schlüsselnamen des Unterarrays gebildet wurde. Die Felder werden nur dort mit Anführungszeichen umgeben, wo es nötig ist.

So erledigen Sie einen Datei-Upload

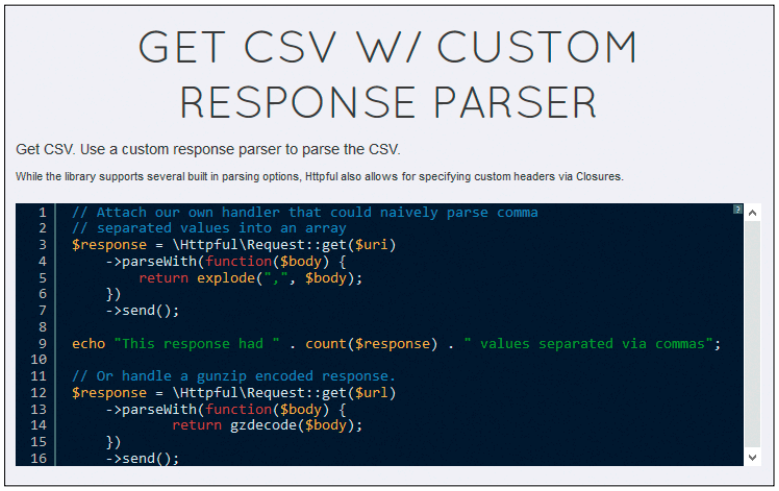
Das Versenden einer Datei über einen File-Upload passiert ja im Browser über ein spezielles Formularfeld und *enctype=multipart/form-data*. Sie müssen sich mit httpful weder um dieses Attribut kümmern noch für das passende Format des Inhalts der gewünschten Datei sorgen.

Diese Arbeiten nimmt Ihnen die Bibliothek ab. Sie nutzen lediglich die Methode *attach()*, die als Parameter ein assoziatives Array mit den Datei-Informationen erwartet. Hier ein Beispiel:

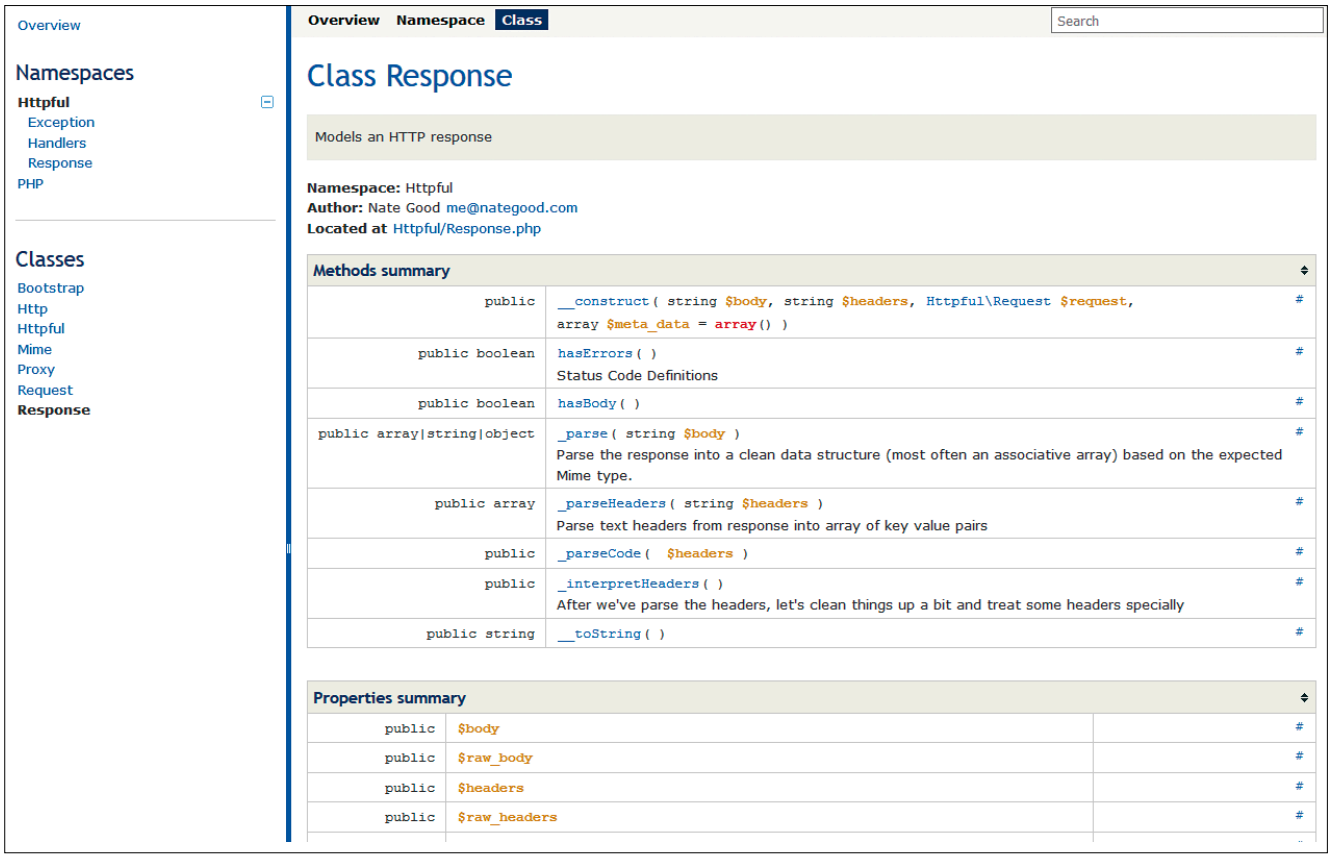
```
$url = "http://myserv.de/upload.php";
$response = \httpful\Request::post($url)
  ->attach(["myfile1"=>"/files/pic.jpg"])
  ->send();
```

Dabei entspricht der Schlüsselname im Array dem Namen, den das Input-Feld im Formular auf der Webseite besitzt. Das obige Skript würde also dieselbe Wirkung haben wie folgende Zeile in einem Upload-Formular:

```
<input name="myfile1" type="file">
```



Eigener Parser: httpful bietet die Möglichkeit, einen individuellen Parser zu registrieren (Bild 3)



Die Online-Referenz der Bibliothek erklärt übersichtlich alle Klassen, Methoden und Parameter von httpful (Bild 4)

Wenn Sie mehrere Dateien hochladen möchten, dann erweitern Sie das Array einfach entsprechend:

```
->attach([
  "myfile1"=>"/files/pic.jpg",
  "myfile2"=>"/files/logo.jpg"
])
```

httpful unterstützt einerseits die Basic Authentication, also die Methode, die im Browser zu dem Aufploppen des kleinen Fensters mit den Feldern für Username und Kennwort führt.

Authentifizierung und SSL

Dazu fügen Sie einfach die Methode *authenticateWithBasic()* hinzu, die Login und Passwort als Parameter erwartet. Hier etwa ein JSON-Aufruf mit einer solchen Anmeldung:

```
$response = \httpful\Request::get($url)
  ->sendsJson()
  ->authenticateWithBasic('user', 'pw')
  ->body('{"foo":"bar"}')
  ->send();
```

Aber diese altbekannte Methode ist nicht die einzige Variante, die httpful beherrscht. Über *authenticateWithCert()* können Sie auch Client-seitige Zertifikate verwenden. Dazu werden Name und Pfad des Key-Files und des Cert-Files sowie optional noch ein Kennwort übergeben. Ansonsten funktioniert die Verwendung genauso wie mit *authenticateWithBasic()* im vorangegangenen Beispiel.

Falls Sie per HTTPS einen Server ansprechen, ist das einfach mit der Verwendung von *https://* als Protokoll im URL getan. Hat der Server allerdings SSL-Probleme, wie etwa ein selbstausgestelltes oder abgelaufenes Zertifikat, wird httpful mit einem Fehler abbrechen. Um der Bibliothek ein entspannteres Verhältnis zum Thema Sicherheit vorzugeben, verwenden Sie in der Aufrufkette die Methode *withoutStrictSSL()*.

Falls Sie einen Proxy für Ihre Abfragen einsetzen möchten oder müssen, genügt es, beispielsweise *->useProxy("172.16.0.254", 8888)* in Ihre Anfrage einzubauen.

So senden Sie spezielle Header

Wenn Sie dem Server zur Anfrage zusätzliche Header mitgeben wollen, können Sie unter verschiedenen Syntaxvarianten wählen.

Angenommen, Sie möchten bewirken, dass der Header *X-My-Header* mit dem Wert *MyValue* dem Aufruf hinzugefügt wird. Dann bauen Sie in den Aufruf zum Beispiel *->withXMyHeader("MyValue")* ein.

Die Library arbeitet mit Magischen Methoden, holt sich also den angesprochenen Methodennamen, baut ihn zum Headerschlüssel *X-My-Header* um und setzt als Wert den als Parameter genannten String.

Falls Ihnen das zu magisch ist, können Sie auch die Methode *addHeader()* einsetzen, die Schlüssel und Wert als se- ▶

Links zum Thema

- Homepage des Projekts httpful
  http://phphttpclient.com
- Strukturierte API-Dokumentation des Projekts
  http://phphttpclient.com/docs
- GitHub-Repository von httpful
  https://github.com/nategood/httpful
- Alternatives Projekt, das sprachübergreifend arbeitet, aber eine weniger elegante Syntax hat
  http://unirest.io

parate Parameter annimmt und den Schlüssel in seiner fertigen Variante erwartet. Folgende Variante wäre also gleichwertig zur Magischen Methode:

```
->addHeader("X-My-Header","MyValue")
```

Mit Hilfe der Methode *addHeaders()* können Sie auch mehrere Header auf einmal setzen. Diese übergeben Sie dann als assoziatives Array:

```
->addHeaders([
  "X-My-Header1","foo"),
  "X-My-Header2","bar")
])
```

Es gibt Fälle, wo Sie selbst einen Parser schreiben möchten, etwa, wenn Sie auf einen Inhaltstyp stoßen, der nicht abgedeckt ist, oder wenn der Datenlieferant von der unterstützten Syntax abweicht.

So integrieren Sie einen eigenen Parser

Dazu bietet httpful die Möglichkeit, einen individuellen Parser zu registrieren. Der muss als Klasse gestaltet sein, die von *MimeHandleAdapter* abgeleitet ist:

```
class MyCsvHandler extends
\httpful\Handlers\MimeHandlerAdapter{
  ...
}
```

Damit Ihre Variante von httpful verwendet wird, registrieren Sie sie unter dem gewünschten MIME-Typ:

```
\httpful\httpful::register('text/csv',
new MyCsvHandler());
```

Die mitgelieferten Beispiele enthalten genau einen solchen Fall. Sie finden den Code im Unterverzeichnis *vendor/nategood/httpful/examples*. Manche Sonderwünsche beherrschen allerdings schon die eingebauten Parser. Möchten Sie beispielsweise bei einer JSON-Antwort die Daten lieber als assoziatives Array erhalten, können Sie dies über einen Zusatzparameter dem normalen Handler mitteilen. Dazu wird er nach folgendem Muster neu registriert:

```
\httpful\httpful::register(
  'application/json', new \httpful\Handlers\JsonHandler(
    array('decode_as_array' => true))
);
```

Die Dokumentation von httpful gibt noch einige Beispiele für einen anderen Ansatz für eigenes Parsen. Dabei wird die Methode *parseWith()* verwendet, die als Parameter eine Closure, also eine anonyme Funktion, erwartet, die die Daten parst (Bild 3).

So legen Sie Vorlagen für Abfragen an

Wenn Sie im selben Skript mehrere Abfragen zum selben Server durchführen müssen, ist es praktisch, die Template-Funktion von httpful zu nutzen. Dabei setzen Sie einmalig die gewünschten Vorgaben, und jeder neue Request verwendet diese dann. Das Template wird zum Beispiel so definiert und bekannt gemacht:

```
use httpful\Request;
$template = Request::init()
  ->withoutStrictSsl()
  ->expectsXml()
  ->sendsXml();
Request::ini($template);
```

Um die Vorgabe zu nutzen, schreiben Sie wie gewohnt:

```
$response = request::get($uri)
  ->body($daten)
  ->send();
```

Nur werden eben alle zuvor gemachten Vorgaben für SSL und den Inhaltstyp XML übernommen. Möchten Sie eine der Vorgaben für einen Aufruf überschreiben, fügen Sie die entsprechende Methode in den Aufruf ein. Das Template bleibt davon unberührt.

Fazit

httpful bietet eine objektorientierte Abstraktion für die Funktionen der libcurl-Library von PHP (Bild 4). Durch eine saubere OOP-Strukturierung und praktikable Syntax kommt man mit httpful schnell zum Ergebnis. ■

Markus Schraudolph
ist Software-Entwickler, Autor von Fachbüchern und Berater. Seine Schwerpunkte sind Webentwicklung und Webtechnologien.

Impressum

Verlag
Neue Mediengesellschaft Ulm mbH
Bayerstraße 16a,
80335 München
Telefon: (089) 7411 7-0,
Fax: (089) 7411 7-101
(ist zugleich Anschrift aller Verantwortlichen)

Herausgeber
Dr. Günter Götz

Chefredakteur
Max Bold
– verantwortlich für den redaktionellen Teil –
E-Mail: redaktion@webundmobile.de

Schlussredaktion
Ernst Altmannshofer

Redaktionelle Mitarbeit
Philip Ackermann, Osvaldo Aguilar, Daniel Basler, Christian Bleske, Siegfried Bolz, Ekkehard Gentz, Tam Hanna, Johannes Hoppe, Bernhard Lauer, Patrick Lobacher, Florence Maurice, Stephan Pohl, Michael Rohrlich, Michael Schams, Jochen Schmidt, Markus Schraudolph, Alexander Schulze, Katharina Sckommodau, Norbert Sendetzky, Thomas Sillmann, Alexander Steireif, Gregor Woiwode

Art Directorin
Maria-Luise Sailer

Grafik & Bildredaktion
Alfred Agatz, Dagmar Breitenbauch, Catharina Burmester, Hedi Hefele, Manuela Keller, Simone Köhnke, Cornelia Pflanzer, Petra Reichenspurner, Ilka Rüther, Christian Schumacher, Nicole Üblacker, Mathias Vietmeier

Anzeigenberatung
Jens Schmidtmann, Anzeigenleiter
Klaus Ahlering, Senior Sales Manager
Telefon: (089) 7411 7-125
Fax: (089) 7411 7-269
E-Mail Anzeigenberatung: sales@nmg.de

Anzeigendisposition
Dr. Jürgen Bossmann
Telefon: (089) 7411 7-281
Fax: (089) 7411 7-269
E-Mail: sales@nmg.de

Leitung Herstellung/Vertrieb
Thomas Heydn
Telefon: (089) 7411 7-111
E-Mail: thomas.heydn@nmg.de

Leserservice
Hotline: (089) 7411 7-205
Fax: (089) 7411 7-101
E-Mail: leserservice@nmg.de

Kooperationen
Denis Motzko
Telefon: (089) 7411 7-116
E-Mail: kooperationen@nmg.de

Druck
L.N. Schaffrath Druckmedien
Marktweg 42–50
47608 Geldern

CD-Produktion
Stroemung GmbH

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.

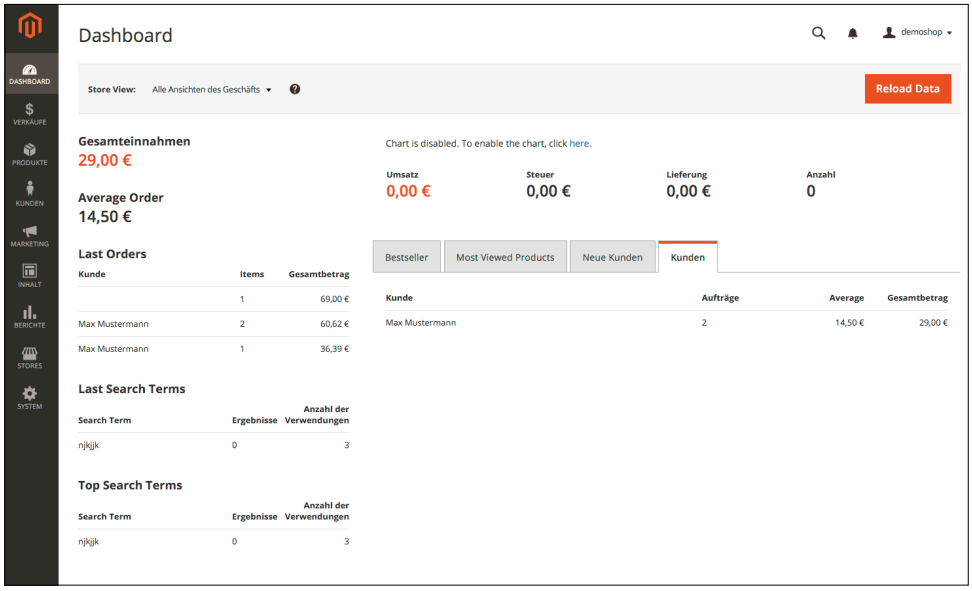
ISSN: 2194-4105

E-COMMERCE-SOFTWARE MAGENTO 2

Evolution im Detail

Es gibt eine neue, komplett überarbeitete Version der populären E-Commerce-Lösung.

Magento stand in den letzten Wochen und Monaten nicht wegen der Software und der seit Langem angekündigten neuen Version im Fokus vieler Diskussionen. Schlagzeilen wurden in erster Linie wegen des Verkaufs von Ebay Enterprise gemacht, jenes Businessbereichs innerhalb des Ebay-Konzerns, zu dem Magento zählt. Zukunftsdiskussionen wurden in erster Linie aufgrund der Unternehmensstruktur und des neuen Eigentümers geführt, nicht aber wegen der Software Magento und deren Zukunftsaussichten.



Alles neu: Magento hat für die Version 2 das Backend komplett umgestaltet (Bild 1)

Dies ist umso ärgerlicher, als mit Magento 2 in wenigen Monaten eine neue Version der populären E-Commerce-Lösung veröffentlicht wird, die komplett mit ihrem Vorgänger bricht. Es wird der erste harte Schnitt in der Historie von Magento werden: Abwärtskompatibilität, ein komplett neuer technologischer Unterbau und eine ganz neue Art und Weise, Module und Themes zu entwickeln, sind die eigentlichen Themen, mit denen sich Magento-Shop-Betreiber, Front- sowie Backend-Entwickler und E-Commerce-Entscheider beschäftigen müssen.

Doch worin liegen letztendlich die Unterschiede und Neuerungen im Detail, und wird Magento 2 wirklich so gut werden, wie viele Magento-Enthusiasten es sich erhoffen?

Die lange Historie von Magento 2

Magento 2 als Vaporware zu bezeichnen, wäre an dieser Stelle vielleicht etwas zu vermessen. Aber das spielt speziell für die Betrachtung der verwendeten Technologien eine wichtige Rolle. Es ist unbestritten, dass Magento 2 mit vielen Verschiebungen zu kämpfen hatte und dadurch an der ein oder anderen Stelle technologisch etwas hinterherhinkt.

Erstmalig angekündigt wurde Magento 2 im Jahr 2010, also vor mehr als fünf Jahren. Ein erster anvisierter Veröffentlichungstermin wurde nach Ankündigung der Version 2 mit Ende 2011 angegeben. Doch dann kam genau in diesem Zeitraum Ebay und kaufte kurzerhand Magento auf, und es entstand, zumindest gefühlt, eine Entschleunigung, speziell in Bezug auf die angekündigte Version 2.

Nach etlichen Verzögerungen, Verschiebungen und Umplanungen hatte man als groben Release-Termin das Ende 2014 bekanntgegeben. Wie man heute weiß, handelte es sich auch hierbei um ein Datum, das nicht gehalten werden konnte. Im Jahr 2015 nahm das Veröffentlichungsdatum von Magento 2 konkretere Züge an, eine erste Beta-Version steht bereits auf GitHub (https://github.com/magento) bereit, und sollte nichts mehr dazwischenkommen, wird Magento 2 Ende 2015 endlich das Licht der Welt erblicken.

Zwischen der Ankündigung und der finalen Veröffentlichung von Magento 2 werden über fünf Jahre liegen, eine enorm lange Zeit im E-Commerce. Diese Verzögerungen können für die Zukunft von Magento zwei unterschiedliche Gefahren bedeuten:

Zum einen steckt in Magento 2 Technologie, die heute nicht mehr ganz als State of the Art anzusehen ist oder zu der einfach bessere Alternative bereitstehen. Dazu zählt ganz konkret die Nutzung des Zend Frameworks in der Version 1. Hierüber sind bereits im Internet Diskussionen entbrannt, die das Für und Wider thematisieren. Dennoch: Zieht sich die Entwicklung einer Software über einen zu langen Zeitraum hin, wird man zwangsläufig Gefahr laufen, auf Technologien gesetzt zu haben, die nach dem Veröffentlichungstermin bereits Nachfolger haben oder gegenüber denen sich schlicht und einfach bessere Alternativen hervorgetan haben.

Neben dem technologischen Aspekt haben die Macher von Magento Vertrauen zerstört, das in letzter Zeit wieder mühe-

voll aufgebaut wird. Letztendlich wissen Shop-Betreiber seit Jahren nicht exakt, wann Magento 2 veröffentlicht wird. Bei der Wahl einer E-Commerce-Plattform spielt die Planungssicherheit jedoch eine äußerst wichtige Rolle. In Projekten im großen sechsstelligen Bereich – also gerade diejenigen, die Magento eigentlich adressieren möchte – ist es von Bedeutung, ob in wenigen Monaten eine komplett neue Version der Software veröffentlicht wird oder eben nicht. Denn die Planungssicherheit in Projekten ist zum Teil auch ein Kriterium für die Wahl des Software-Anbieters. Hier bestand in der Vergangenheit definitiv ein Mangel in der Transparenz und Kommunikation.

Speziell aus diesen Gründen sind es vielleicht nicht die besten Voraussetzungen, die Magento 2 mit sich bringt. Die Historie kann auch nicht bereinigt werden. Was jedoch zu sehen und zu spüren ist, sind die Bemühungen von Magento, die Fehler der Vergangenheit zu korrigieren und hier, speziell in Sachen Kommunikation und Transparenz, einen anderen Weg einzuschlagen.

Das bedeutet Magento 2

Vergleicht man die Entwicklung von Magento 2 mit der mittlerweile populären E-Commerce-Lösung Shopware, so lässt sich ein großer Unterschied feststellen. Shopware setzt seit einigen Versionen konstant auf die Entwicklung neuer Funktionen und Möglichkeiten für Shop-Betreiber, gepaart mit technologischen Anpassungen.

Magento 2 hingegen wird vor allem durch Änderungen in der Code-Struktur und dem Verborgenen glänzen. Oder anders formuliert: Shop-Betreiber, Administratoren und E-Commerce-Entscheider werden bei der Version 2 von Magento voraussichtlich nicht so hoch jubeln, da sich für sie, abgesehen von optischen Anpassungen, relativ wenig ändert. Die Features, die in der momentanen Community- beziehungsweise. Enterprise-Edition vorhanden sind, werden auch in Magento 2 vorhanden sein. Nicht mehr, aber auch nicht weniger.

Das bedeutet aber wiederum: In Magento 2 wird sich eine Menge ändern, vieles ist aber für das Auge gar nicht sichtbar. So setzt Magento 2 auf CSS3, HTML5, LESS, RequireJS, Composer, PHP 5.5, MySQL 5.6, das Zend Framework 1 und jQuery. Die Performance wurde stark optimiert und der Code wird endlich automatisiert testbar sein. Ergänzt wird dies durch eine umfangreiche Dokumentation, die für jedermann verfügbar sein wird. So bringt Magento viele Veränderungen und Weiterentwicklungen, aber diese beziehen sich größtenteils auf das Innenleben.

Ein neues Backend für mehr Übersicht

Die für Shop-Betreiber vermutlich auffallendste Neuerung ist das neu gestaltete Magento-Backend. Der Administrationsbereich erscheint dabei nicht nur in einer gänzlich veränderten Optik, auch die Strukturierung der Menüpunkte wurde komplett überarbeitet. Einziger Wermutstropfen: Das Backend ist in dieser Form auf mobilen Endgeräten wie Tablets nur mit etwas Mühe nutzbar. Ein responsive Backend-Design wäre meiner Meinung nach auf jeden Fall ein interessanter USP gewesen.

Ist das alte Magento an der ein oder anderen Stelle unlogisch aufgebaut, bietet Magento 2 eine wesentlich bessere Übersichtlichkeit.

Dreh- und Angelpunkt des Backends ist die Hauptnavigation, die sich im linken Seitenrand befindet und sich in folgende Punkte untergliedert: *Dashboard, Verkäufe, Produkte, Kunden, Marketing, Inhalt, Berichte, Stores* und *System*.

Der Einstiegspunkt in das Backend ist das *Dashboard*. Dieses ist bereits aus Magento 1 bekannt und bietet für Shop-Betreiber eine schnelle und einfache Übersicht über unternehmenskritische Daten. Dazu zählen beispielsweise die letzten Bestellungen, der aktuelle Umsatz und Informationen zu Kunden und Produkten. Dieser Bereich bietet wenig Neues, auch wenn die Übersichtlichkeit durch das neue Design erhöht wurde (Bild 1).

Im Punkt *Verkäufe* verbirgt sich die gesamte Funktionalität, die für die Bearbeitung von Bestellungen benötigt wird. Dazu zählen das Anlegen von Rechnungen und Lieferscheinen sowie die Bearbeitung von Bestellungen. Glücklicher- ▶

Mitmachen und gewinnen

Die web & mobile developer verlost drei Handbücher für angehende Betreiber eines Online-Shops.

Diesen nützlichen Ratgeber können Sie gewinnen

Wenn Sie einen Online-Shop starten möchten, müssen Sie vor dem Startschuss wichtige Entscheidungen treffen: Welche E-Commerce-Software ist für Ihre Zwecke am besten geeignet? Was müssen Sie bei Versandarten sowie Bezahlungssystemen beachten? Und ganz entscheidend: Was kostet Sie das? Mit dem umfassenden »Handbuch Online-Shop« vom Autor dieses Artikels erhalten Sie alles, was Sie für den Betrieb eines Online-Shops benötigen: von den ersten Schritten über wichtiges Usability- und Marketing-Wissen bis hin zu wertvollen Tipps, damit Sie rechtliche und buchhalterische Fallstricke vermeiden. So stellen Sie sich den vielfältigen Herausforderungen und Trends im E-Commerce.

web & mobile developer verlost drei dieser nützlichen Ratgeber für angehende Betreiber eines Online-Shops. Wenn Sie Interesse haben und eines dieser E-Books gewinnen wollen, schicken Sie einfach eine E-Mail mit dem Betreff »Magento-Verlosung« an redaktion@webundmobile.de. Mit etwas Glück gehören Sie zu den Gewinnern und können sich das E-Book herunterladen oder online lesen. Der Rechtsweg ist ausgeschlossen.

weise wurde der Punkt mit der Steuerberechnung aus dem Verkaufsbereich entfernt, ebenso wie die Verwaltung der Bestellbedingungen. Shop-Betreiber können Bestellungen wie bereits in Version 1 bearbeiten, hier gibt es keinerlei Umstellungen, die den Workflow betreffen.

Der in Magento 1 sehr umfangreiche Punkt *Katalog* wurde komplett aufgetrennt. Der neu entstandene Bereich *Produkte* kümmert sich ausschließlich um den Produktkatalog und bietet darüber hinaus keine weiteren Möglichkeiten, wie beispielsweise die Verwaltung von Kommentaren, Ratings oder Attribute.

Die Verwaltung der Produkte und Kategorien ist, bezogen auf das Handling und den Funktionsumfang, im Prinzip identisch zur Vorgängerversion. Auch hier wurde letztendlich nur an der Optik gearbeitet. Wer in Magento 1 bereits Produkte und Kategorien verwaltet hat, wird auch in der neuen Version damit keine Probleme haben.

Innerhalb der Kundenverwaltung sieht es ähnlich aus. Dieser Menüpunkt birgt, abgesehen von der Verwaltung der Kundengruppen, keine Neuerungen.

Überarbeitet wurde hingegen der Bereich *Marketing*. Neben Preisregeln ist hier ebenso die Verwaltung von E-Mail-Templates und Newslettern angesiedelt. Dieser Menüpunkt beinhaltet alle in Magento vorhandenen Funktionen, die Ihnen als Shop-Betreiber bei der Generierung von Umsätzen weiterhelfen. Hierzu zählen neben den Preisregeln und E-Mail-Templates auch umfangreiche Möglichkeiten in Bezug auf Kundenbewertungen. Aber auch die Verwaltung der URLs ist speziell für das Thema SEO ein wichtiger Punkt.

Ein Feature, welches ich mir seit Jahren wünsche, hat auch in Version 2 keinen Einzug gefunden: ein Wysiwyg-Editor für E-Mails. Wer bereits E-Mail-Templates in Magento erstellt oder editiert hat weiß, dass dies nur mit HTML/CSS möglich ist (Bild 2). Dies stellt speziell technisch unerfahrene Shop-Betreiber vor große Herausforderungen.

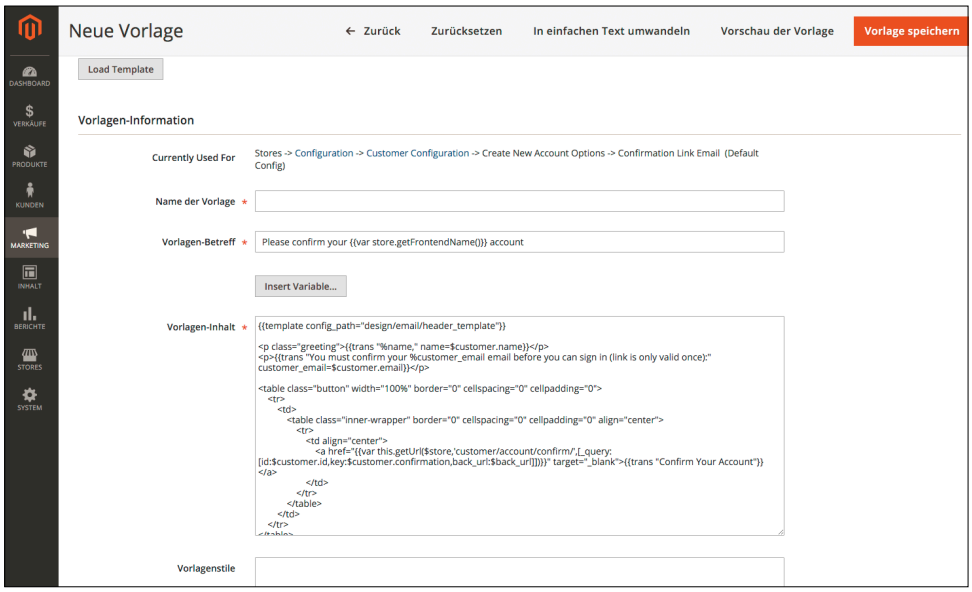
Im nächsten Menüpunkt *Inhalte* sind die Elemente aus dem bereits aus Version 1 bekannten Bereich *CMS* zu finden. Neben den Seiten und statischen Blöcken können innerhalb dieses Bereiches Designänderungen verwaltet werden. Diese Zusammenfassung der Inhaltsbereiche ist definitiv sinnvoll und erhöht die Übersichtlichkeit.

Die Berichte bieten einen Überblick über unternehmenskritische Zahlen. Fairerweise muss man an dieser Stelle erwähnen, dass die Berichte aber letztendlich auch in Version 1 von Magento keine allzu große Rolle spielen. Diese ersetzen definitiv kein Webanalyse-Tool. Man kann sie nutzen, aber man darf hier keine Wunder erwarten.

Spannend hingegen wird es definitiv in den letzten beiden Menüpunkten, *Stores* und *System*. Innerhalb des Menüpunkts *Stores* sind grundlegende Konfigurationsmöglichkeiten für den Online-Shop möglich. An dieser Stelle werden zukünftig Attribute und Kundengruppen verwaltet, Steuerregeln hinterlegt und Konfigurationsoptionen gepflegt. Der Punkt *Stores* löst damit den enorm überfrachteten Punkt *Konfiguration* ein Stück weit ab und bietet eine zentrale Anlaufstelle, innerhalb derer alle Einstellungsmöglichkeiten durchgeführt werden können.

Der Punkt *System,* den man auf den ersten Blick mit der Konfiguration verwechseln könnte, bietet hingegen Funktionen in Sachen Import/Export, Cache- und Index-Management sowie Berechtigungen. Wer bislang in Magento 1 mit dem enorm umfangreichen Punkt *System* und *System*, *Konfiguration* Übersichtlichkeitsprobleme hatte, wird sich als Folge dieser Neuerung definitiv besser zurechtfinden.

Bezogen auf das Backend bringt Magento 2 einen großen Vorteil: Übersichtlichkeit. Diese wurde enorm gesteigert und Verbesserungen und Optimierungen wurden im Detail durchgeführt. Ein neues Design sorgt für einen zeitgemäßen und wesentlich übersichtlicheren Eindruck, Menüpunkte wurden logischer gruppiert und insgesamt wirkt das Backend nicht mehr so stark überfrachtet wie in Version 1. Wer jedoch auf spannende neue Features hofft, wird vom Backend enttäuscht werden. Hier findet definitiv eine Evolution und keine Revolution statt.



Tricky E-Mail Templates: Leider steht weiterhin kein grafischer Editor für Mail-Templates zur Verfügung (Bild 2)

Responsive Frontend 2.0

Neben dem Backend fährt Magento 2 mit einem komplett überarbeiteten und neu gestalten Frontend auf. Hier ist vor allem die Technologie unter der Haube interessant. So setzt Magento standardmäßig auf RequireJS, hat sich komplett von Prototype verabschiedet und setzt auf jQuery. In Sachen CSS verwendet Magento zukünftig LESS. Dieser Punkt sorgt im Übrigen speziell im In-

ternet für Diskussionen, gibt es doch sowohl auf Seiten von Sass wie auch LESS Verfechter. Aufgrund der Stabilität hatte man zum damaligen Entwicklungszeitpunkt von Magento beschlossen, auf LESS zu setzen.

Neben den rein technischen Änderungen tritt Magento 2 mit einem komplett neu entwickelten Standard-Theme namens Luma an. Bei dem Luma-Theme handelt es sich um ein minimalistisches, auf Usability optimiertes Responsive-Design, das sich als Basis für eigene Designentwicklungen oder Modifikationen nutzen lässt. Im Vergleich zum Design des Vorgängers verfügt Luma über einen sehr reduzierten Kopf-Bereich und eine komplett überarbeitete Startseite.

Hier arbeitet das Design standardmäßig mit einer Kacheloptik und stützt sich dadurch primär auf in dem Vordergrund stehende Grafiken. Auf Slider wurde gänzlich verzichtet. Alle Inhaltselemente sind durch Scrollen ersichtlich, es gibt keine versteckten oder verdeckten Inhalte, die man erst aufklappen oder sliden muss. Die Kacheloptik eignet sich speziell für mobile Endgeräte, da hier flexibel mit der Anordnung beziehungsweise Darstellung der Kacheln gespielt werden kann. Abgeschlossen wird die Seite mit einer Darstellung von Produkten sowie einem Footer (Bild 3).

Innerhalb der Kategorieansicht hat sich, abgesehen von der Optik, nicht viel geändert. Schade, denn bei der Filternavigation hat man exakt dasselbe Problem, das schon seit Magento 1 besteht: Wenn Sie innerhalb einer Kategorie eine große Anzahl an Produkten darstellen, möchten Ihre Kunden in der Regel die Ergebnisse filtern, beispielsweise nach dem Preis, einer Größe oder Farbe. Im Modebereich ist es nicht unüblich, nach mehreren Farben zu filtern – gegebenenfalls möchte Ihr Kunde schließlich alle schwarzen und dunkelblauen Produkte sehen.

Doch genau diese Mehrfachauswahl pro Eigenschaft ist leider weiterhin nicht möglich und muss, wie auch in Version 1, nachgerüstet werden. Speziell bei solchen Funktionen hätte ich mir persönlich eine sinnvolle Weiterentwicklung des Standard-Funktionsumfangs gewünscht.

Spannend und damit auch funktional eine der wenigen Neuerungen ist der Bezahlvorgang. Denn dieser erfolgt, gänzlich nach dem Muster von Amazon in einem sehr reduzierten Design. Der Kopf- und Fußbereich, der auf jeder vorherigen Seite zu sehen ist, verschwindet gänzlich und der Online-Shop-Kunde kann seine Transaktion vollkommen ungestört von Ablenkungen durchführen.

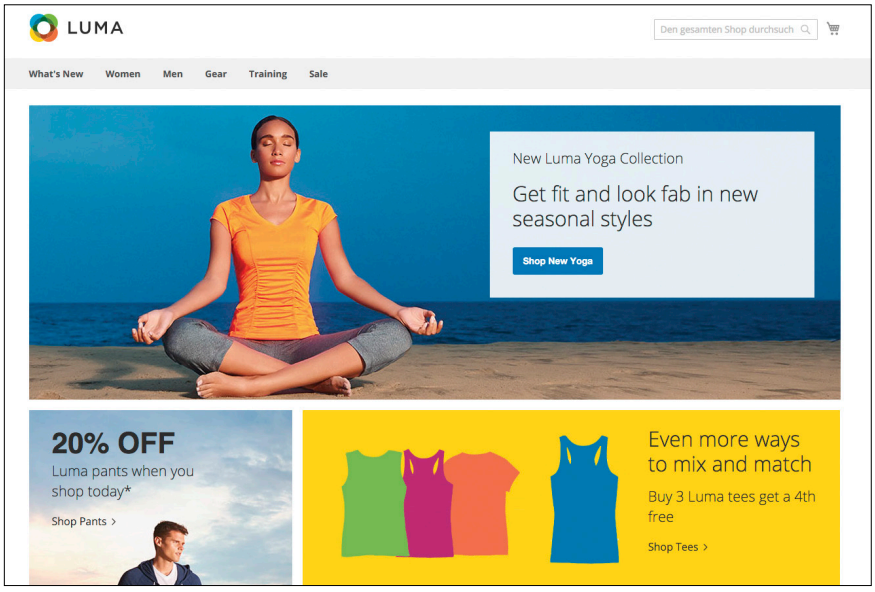
Tatsächlich wurde der bislang als kompliziert titulierte und oftmals durch Erweiterungen abgelöste Standard-Checkout sinnvoll durch ein vom Design und von der Funktionalität her bessere Variante ersetzt, die dem Besucher ein schnelles Einkaufen ermöglicht (Bild 4).

Zusammengefasst bietet das neue Magento-Standard-Theme namens Luma logische (Weiter-)Entwicklungen und erscheint als State-of-the-Art-Design, mit dem man als Basis definitiv arbeiten kann.

Unter der Haube gibt es eine ganze Reihe sinnvoller Ergänzungen und Anpassungen. Insbesondere der Checkout ist eine sehr gute Weiterentwicklung. Revolutioniert wurde auch an dieser Stelle sicherlich nichts, aber speziell für den schnellen Start eines E-Commerce-Projekts stellt das neue Theme eine gute Basis dar.

Modul- und Template-Entwicklung

Shop-Betreiber werden durch das neue Backend auf ihre Kosten kommen. Verwendet man das neue Luma-Theme, tut man den eigenen Kunden etwas Gutes. Aber was bringt Magento 2 für Front- und Backend-Entwickler?



Luma-Theme: Das neue Design ist responsive und minimalistisch gehalten (Bild 3)

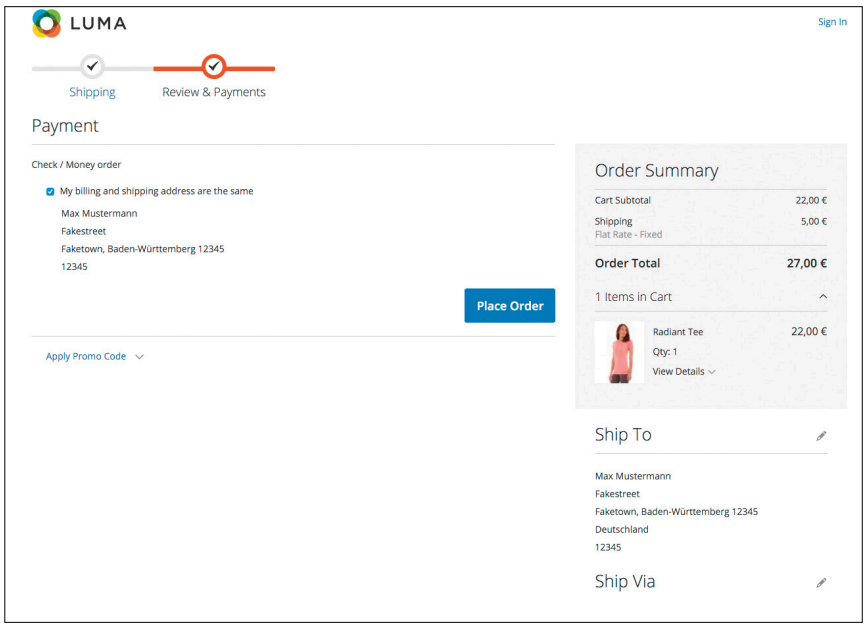
Die gute Nachricht zuerst: Magento versucht, mit einer umfangreichen Dokumentation sowohl Frontend- als auch Backend-Entwickler zu unterstützen. Das leidige Thema Dokumentation spielt vor allem in Version 1 eine große Rolle. Unter http://devdocs.magento.com findet sich bereits umfangreiches Informationsmaterial zur neuen Magento-Version 2. Hier hat Magento definitiv aus den Fehlern der Vergangenheit gelernt und ist gewillt, den Entwicklern alle nötigen Informationen an einer zentralen Stelle bereitzustellen.

Für Frontend-Entwickler, die sich auf die Realisierung von Themes konzentrieren, ändert sich mit Version 2 zuallererst die Struktur bei der Theme-Entwicklung. Die Trennung zwischen *app/design* und dem *skin*/-Verzeichnis wurde komplett abgeschafft. Als zentrale Anlaufstelle dient, bezogen auf das Frontend-Design, das Verzeichnis *app/design/frontend*. Innerhalb dieses Verzeichnisses wird ein Hauptverzeichnis und innerhalb dessen ein weiteres Unterverzeichnis erstellt, zum Beispiel *app/design/frontend/meindesign/default*. Diese ▶

Trennung, in Magento 1 nannte es sich Package und Theme, sollte für Frontend-Entwickler bereits bekannt sein. Neu ist hingegen die Struktur innerhalb des eigenen Design-Verzeichnisses. So muss das Theme eine Struktur aufweisen, wie sie in Bild 5 zu sehen ist. Tabelle 1 zeigt, welche Bedeutung die einzelnen Verzeichnisse haben.

Zu guter Letzt existieren zwei Definitionsdateien, die man benötigt, um das Theme zur Verfügung zu stellen. Dabei handelt es sich um die *registration.php* und die *theme.xml*.

Grundsätzlich haben sich bei der Entwicklung des Themes in erster Linie die Struktur und der Aufbau geändert. Man hat die Möglichkeit, wesentlich modularer zu arbeiten, die Struktur wurde dahingehend optimiert, dass das */skin*-Verzeichnis nicht mehr notwendig ist, und insgesamt ist der neu entwickelte Aufbau definitiv sinnvoller. Die Art und Weise, ein Design aufzubauen, ist im direkten Vergleich zur Vorgängerversion im Prinzip identisch. So gibt es Layout-XML-Files, die definieren, an welcher Stelle welches Element geladen wird. Weiterhin gibt es klassische Template-Files mit einer Mischung aus HTML- und PHP-Code sowie CSS- und JavaScript-Dateien. Frontend-Entwickler müssen sich daher auf Änderungen gefasst machen, aber nicht komplett von null an beginnen.



Revolution im Checkout: Der neu gestaltete Checkout sorgt sicherlich für Steigerungen der Conversion-Rate (Bild 4)

Für Backend-Entwickler, die Module entwickeln und somit die Funktionalität von Magento erweitern oder ergänzen, hat sich eine ganze Menge geändert. Letztendlich bringt Magento 2 speziell für die Entwickler viele neue Änderungen und Möglichkeiten, und damit sind Entwickler auch diejenigen, für die Magento 2 die größte Umstellung bedeutet.

Zwar wird weiterhin auf das Zend Framework in der Version 1 gesetzt, dennoch hat sich der Unterbau von Magento 2 im Vergleich zu seinem Vorgänger komplett verändert. Die erste auffällige Neuerung ist die geänderte Struktur im Bereich *app/code*. Während in Magento 1 eine Untergliederung nach *core, community* und *local* existiert, wurde diese in Version 2 entfernt.

Der Aufbau eines Moduls unterscheidet sich ebenfalls stark von der vorherigen Version. So gibt es in Magento 2 innerhalb eines Moduls Controller, Blöcke, XML-Definitionsdateien sowie Template-Files und auch Übersetzung-Files. Hier entsteht tatsächlich eine komplett neue Struktur bei der Modulentwicklung.

Die hier resultierenden komplexen Änderungen können Sie sich idealerweise durch einen Blick in die Dokumentation von Magento 2 unter http://devdocs.magento.com/guides/v2.0/extension-dev-guide/build.html im Detail ansehen. Auch wenn sich durch die neue Struktur und Methodiken bei der Modulentwicklung die Komplexität erhöht, bietet Magento 2 auf der anderen Seite eine verbesserte Möglichkeit, Code automatisiert zu testen und grundsätzlich robustere Erweiterungen zu entwickeln.

Tabelle 1: Struktur des Design-Verzeichnisses

<Vendor>_<Module>/	Hierin befinden sich modulspezifische Inhalte wie Template- und Layout-Dateien. Auch werden CSS-Dateien im .css- beziehungsweise .less-Format innerhalb dieses Verzeichnisses abgelegt. Der Sinn und Zweck besteht darin, für jedes Modul gekapselte Dateien in einem eigenen Modulordner zu verwalten.
etc/	In diesem Verzeichnis kann eine XML-Datei hinterlegt werden, die unter anderem Konfigurationseinstellungen beinhaltet.
I18n/	Übersetzungen werden wie gewohnt im .csv-Format hinterlegt. Neu ist hingegen der Speicherort. Denn Übersetzungen, die Sie für Ihr Design nutzen möchten, werden nun im Ordner i18n abgelegt.
/media	Hierin kann eine Preview des Designs abgelegt werden
/web	Das /web-Verzeichnis ist das große Sammelbecken für CSS- und JavaScript-Dateien sowie Grafiken. Alle statischen Bestandteile Ihres Designs können im /web-Verzeichnis abgelegt werden. Dieses Verzeichnis spiegelt daher den aus Magento 1 bekannten Skin-Ordner zum Teil wider.

Dank der Veränderung der technologischen Basis werden darüber hinaus beachtliche Performance-Steigerungen erzielt. Teilweise ist von Steigerungsraten von bis zu 25 Prozent die Rede. Auch die Skalierbarkeit wurde verbessert, was Magento im Einsatz bei großen und komplexen Projekten einen Vorteil verschafft.

Bezogen auf den technologischen Unterbau kann man bei Magento 2 tatsächlich von einer kleinen Revolution sprechen. Denn unter der Haube hat sich tatsächlich viel geändert. Sind die neuen Möglichkeiten im Bereich der Frontend-Entwicklung noch halbwegs überschaubar, so bietet Magento bei der Modulentwicklung komplett neue Möglichkeiten. Die neuen Möglichkeiten sorgen jedoch auch für eine größere Komplexität und erfordern im ersten Schritt viel Einarbeitungszeit. Die Modulentwicklung nimmt definitiv an Komplexität zu, und speziell in diesem Bereich bewegt sich Magento 2 stark in Richtung Enterprise-System.

```
app/design/frontend/<Vendor>/<theme>/
├── <Vendor>_<Module>/
|       ├── web/
|       |       ├── css/
|       |       |       ├── source/
|       ├── layout/
|       |       ├── override/
|       ├── templates/
├── etc/
├── i18n/
├── media/
├── web/
|       ├── css/
|       |       ├── source/
|       ├── fonts/
|       ├── images/
|       ├── js/
├── composer.json
├── registration.php
├── theme.xml
```

Theme-Struktur: Mit Version 2 ändert sich unter anderem die Theme-Struktur (Bild 5)

Was passiert eigentlich in Magento 1?

Neben den Diskussionen über die neuen Möglichkeiten und Funktionen von Magento 2 muss man definitiv auch über die Zukunft von Magento 1 nachdenken. Denn eines ist auch klar: Sobald Magento 2 veröffentlicht wird, wird es noch Monate dauern, bis populäre Extensions für Magento 2 verfügbar sein werden.

Hierbei geht es um für Shop-Betreiber definitiv kritische Funktionserweiterungen wie Payment-Extension, Erweiterungen für den Versand, Schnittstellen und Co. Auch wenn aktuell im Hintergrund bei vielen Unternehmen schon intensiv an Magento-2-Erweiterungen gearbeitet wird, so muss man realistischerweise auch sagen, dass zum Launch gar nicht alles bereitgestellt werden kann. Auch werden Online-Shops, die über die letzten Jahre hinweg modifiziert und erweitert wurden, nicht von heute auf morgen auf Magento 2 umsteigen können.

Die Migration von Magento 1 auf Magento 2 wird, so ist zumindest der aktuelle Stand, definitiv nicht per Knopfdruck möglich. Daten müssen migriert und Extensions angepasst werden – kurz gesagt wird eine Migration mit viel Aufwand verbunden sein. Magento ist sich dieser Problematik bewusst, weswegen die Version 1.x noch drei Jahre nach dem Launch von Magento 2 unterstützt wird.

Shop-Betreiber können aus diesem Grund auf der einen Seite aufatmen, denn eine Migration muss nicht von heute auf morgen durchgeführt werden. Auch ist der Zeitraum von drei Jahren ausreichend, speziell im Bezug auf einen kompletten Relaunch.

Für Entwickler, Designer und Agenturen wird es hingegen eine spannende Zeit werden, wenn parallel Magento-1- und Magento-2-Projekte realisiert und unterstützt werden müssen. Denn hier werden die Anforderungen an Entwickler definitiv stark steigen.

Fazit

Revolution oder Evolution, das ist hier die Frage. Pauschal lässt sich diese jedoch in Bezug auf Magento 2 nicht beantworten. Viel mehr wird man mit Magento 2 eine Mischung aus beidem vorfinden. Faktisch wird es keine großartigen neuen Funktionalitäten geben. Features werden identisch zum Vorgänger vorhanden sein, das Backend macht einen aufgeräumten und überarbeiteten Eindruck und das neue Standard-Theme ist State of the Art. Hier kann man definitiv von einer Evolution sprechen. Dafür hat sich im Hintergrund im Großen und Ganzen alles geändert, was Anforderungen an Frontend- und Backend-Entwickler stellen wird. Speziell was die Modulentwicklung angeht, hat man wohl mit einer kleinen Revolution zu kämpfen, doch sind die neuen gewonnenen Möglichkeiten sehr vielseitig.

Magento liefert mit der Version 2 ein gutes und solides Produkt, das über viele Verbesserungen und Optimierungen verfügt. Die große Feature-Schlacht, wie sie etwa bei neuen Versionen von Shopware gefeiert wird, bleibt hingegen aus. Das macht Magento 2 aber nicht per se schlechter, sondern verstärkt eher den Enterprise-Aspekt, denn mit Magento 2 wollte man vor allem einen guten, stabilen und skalierbaren Unterbau erschaffen, dessen Funktionsumfang primär von Agenturen und deren Entwicklern erweitert wird. ■

Links zum Thema

- Magento-2-Dokumentation für Entwickler
  http://devdocs.magento.com
- Magento 2 auf GitHub
  https://github.com/magento/magento2
- Weiterführende Informationen und Links zu Magento 2 (inoffiziell)
  https://firebearstudio.com/blog/magento-2-uberblick-funktionen-reviews-tutorials-demo-marz-2015.htm

Alexander Steireif
ist Gründer und Mitinhaber der Magento Agentur ITABS. Er ist auf das Thema E-Commerce spezialisiert und setzt dabei primär auf die E-Commerce-Plattform Magento.
www.alexander-steireif.com

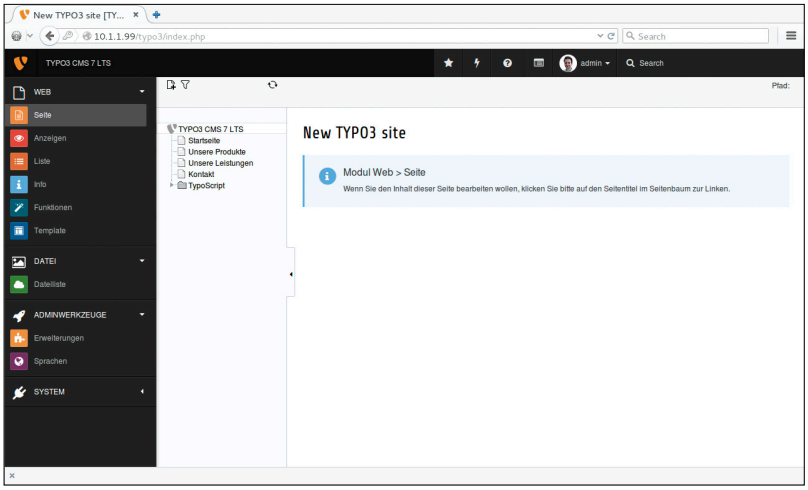
TYPO3 CMS 7 LTS

Moderne Technologien

Vom bekannten CMS TYPO3 wurde eine neue LTS-Version veröffentlicht.

Abgesehen von einem überarbeiteten Backend und vielen Neuerungen unter der Haube bietet die neue Version eine Vielzahl von Verbesserungen für Systemintegratoren und Entwickler, setzt aber auch eine moderne Serverumgebung voraus.

Generell wurde der Kern in vielerlei Hinsicht überarbeitet und auf einen fortschrittlichen und stabilen Stand gebracht, der TYPO3 CMS fit für die nächsten Jahre macht. Schließlich handelt es sich um eine LTS-Version, die mindestens drei Jahre lang Maintenance, Bugfixes und Security-Updates erhalten wird.



Das Backend von TYPO3 CMS 7 LTS (Bild 1)

Agile Release Cycle

Etwas gewöhnungsbedürftig ist die Versionsnummer 7 LTS, wenn man bedenkt, dass im Verlauf der letzten Monate bereits einige Versionen im 7.x-Zweig freigegeben wurden. TYPO3-CMS-Versionen, die den LTS-Status genießen, geben besonders Agenturen, aber auch Entwicklern die Sicherheit, auf eine stabile Code-Basis zu setzen. Diese wird über einen besonders langen Zeitraum unterstützt. Die Resonanz auf das LTS-Konzept, das bereits 2011 mit TYPO3 Version 4.5 eingeführt wurde, ist erwartungsgemäß sehr positiv ausgefallen und auch in anderen Projekten etabliert (zum Beispiel bei Ubuntu Linux, Drupal, Node.js und anderen).

Neue Features und moderne Technologien erfordern ausgiebige Tests und werden daher in LTS-Versionen nur zeitverzögert eingeführt. Unter anderem aus diesem Grund wurde im November 2014 ein neues Release-Konzept angekündigt. Bei TYPO3 CMS werden diese in Sprint-Releases – also in Versionen zwischen LTS-Versionen – implementiert, die in relativ kurzen Abständen veröffentlicht werden.

Somit können mögliche Probleme frühzeitig identifiziert und korrigiert werden.

Sprint-Releases besitzen einen sehr schnellen Lebenszyklus. Sobald eine neue Version veröffentlicht wird, ist die vorherige veraltet und wird nicht mehr weiter unterstützt. Ein Update von einem Sprint-Release zum nächsten ist daher empfehlenswert und funktioniert in den meisten Fällen reibungslos.

Obwohl LTS-Versionen drei Jahre lang mit Updates versorgt werden, wird die nächste LTS Version von TYPO3 CMS bereits in circa eineinhalb Jahren erscheinen. Durch diese Versionspolitik (Agile Release Cycle genannt) ist es sogar theoretisch möglich, eine Version zu überspringen, wenn man Instanzen von LTS- zu LTS-Version aktualisiert.

Angefangen von der Version 7.0, die im Dezember 2014 veröffentlicht wurde, bis einschließlich Version 7.5 (Oktober 2015) wurden insgesamt sechs Sprint Releases von TYPO3 CMS der Öffentlichkeit vorgestellt. Die neue Version stellt nun den Abschluss der 7.x-Serie dar und trägt den Namen TYPO3 CMS 7 LTS.

Überarbeitetes Backend

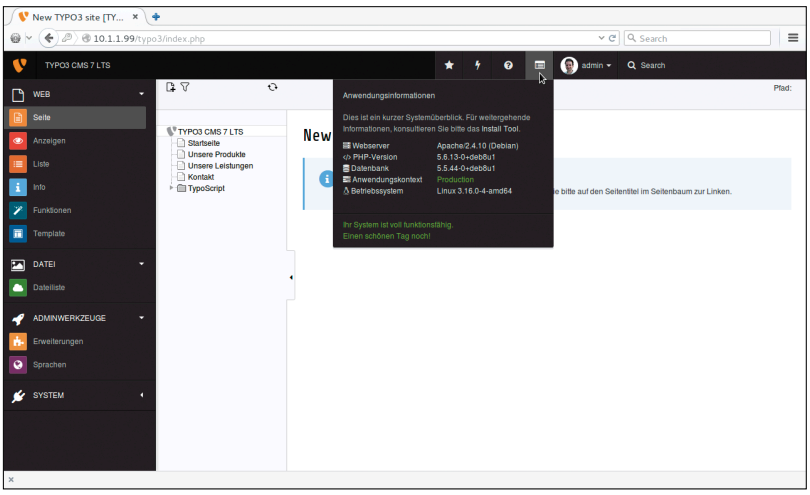
Abgesehen von einem neuen Anmeldeformular, das sich in einigen Details individuell anpassen lässt (siehe Anleitung im Kasten), fällt als Erstes ein überarbeitetes Backend auf. Die neue Version von TYPO3 CMS basiert auf dem Twitter-Bootstrap-Framework und ist somit in vielen Aspekten auch auf mobilen Endgeräten gut bedienbar.

Generell setzt das Backend auf ein Flat-Design mit farbigen und aussagekräftigen Icons. Wie bei bisherigen TYPO3-Versionen üblich, findet man diese überwiegend in der linken Funktionsleiste. Diese Leiste kann so weit minimiert werden, dass nur noch die Icons ohne beschreibenden Text sichtbar sind (Bild 1). Redakteure werden diese Funktion bald schätzen lernen, da man die Bedeutung der Icons sehr schnell auswendig kennt und die Minimierung den Arbeitsbereich auf der rechten Seite deutlich vergrößert.

Die Icons nutzen das Schrift- und CSS-Toolkit Font Awesome, das bekannt für seine skalierbaren Vektor-Icons ist und auch auf hochauflösenden Retina-Displays für eine perfekte Darstellung sorgt. Das einheitliche Design der Icons erstreckt sich über das gesamte Backend: angefangen von der bereits erwähnten Funktionsleiste über sämtlichen Listen bis hin zu internen Modulen.

Einige Module wurden aus der Funktionsleiste entfernt. Wer auf den ersten Blick die Benutzereinstellungen oder die Hilfefunktionen vermisst, findet diese nun in der oberen Leiste wieder. Für Administratoren bietet ein neuer Menüpunkt einen schnellen Überblick über das aktuelle System (unter anderem Webserver, Datenbankserver und PHP-Version) und weist auf mögliche System- und Konfigurationsprobleme hin (Bild 2). Systemintegratoren und Entwickler können dieser Systemübersicht eigene Informationen hinzufügen.

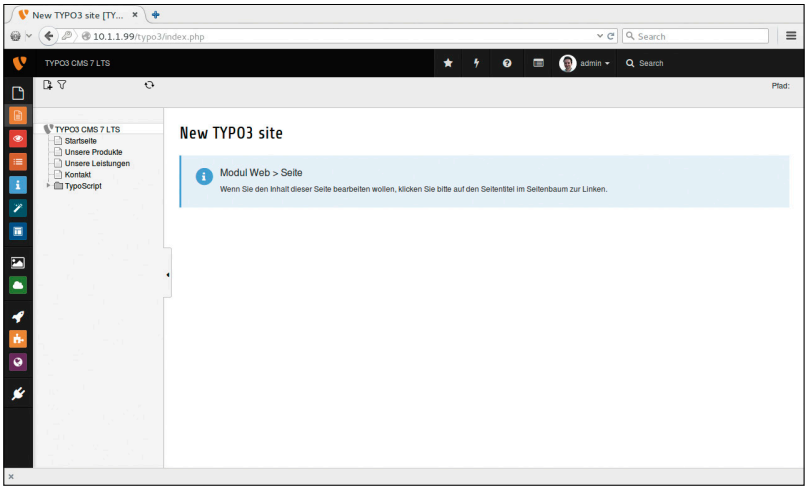
An mehreren Stellen im Backend können Benutzer Datums- beziehungsweise Zeitangaben machen. Das betrifft beispielsweise die Konfiguration, zu welchem Zeitpunkt eine Seite, ein Inhaltselement oder ein News-Artikel automatisch veröffentlicht oder wann ein Task im Planer (Scheduler) ausgeführt werden soll. Diese Auswahl wird nun mit einem Bootstrap-kompatiblen Date-Picker realisiert, der sich ideal in das neue Backend-Design einfügt.



Kurzer Systemüberblick für Administratoren (Bild 3)

Zu den eher kleinen, aber dennoch nützlichen Verbesserungen im Backend gehört unter anderem eine neue Funktion, bestimmte Datensätze mit Beschreibungen auszustatten. Diese beinhaltet derzeit zum Beispiel Dateispeicher (Filemounts), Inhaltselemente (Content Elements) und sogar Backend-Benutzer.

Zusätzlich wird während der Eingabe von Daten in verschiedenen Textfeldern die Anzahl der verbleibenden Zeichen angezeigt. Diese und andere Funktionen sind einer vollständigen Runderneuerung der sogenannten Form-Engine von TYPO3 CMS zu verdanken. Auch wenn diese Arbeit für Redakteure nicht sofort sichtbar ist, ist die Form-Engine eine zentrale und wichtige Komponente, da sie für das Rendern sämtlicher Formulare im Backend verantwortlich ist. Die überarbeitete Version ist nicht nur robuster und auf dem aktuellen Stand der TYPO3-Technik, sondern erlaubt es Extension-Entwicklern auch, eigene Felder zu implementieren und deren Aussehen zu beeinflussen.



Die minimierte Funktionsleiste zeigt lediglich Icons und bietet viel Platz (Bild 2)

Auf eine kleine Umgewöhnung müssen sich Redakteure bei der Auswahl von Inhaltselementen einstellen. Als typischen Seiteninhalt gab es bisher unter anderem die Elemente *Text*, *Text und Bilder* und *Nur Bilder*. Ein neues Inhaltselement ersetzt seit TYPO3 Version 7.5 die beiden erstgenannten. Hierbei handelt es sich um *Text & Media*.

Inhaltselemente und Bildbearbeitung

Wie der Name bereits vermuten lässt, unterstützt dieses neue Inhaltselement nicht nur normale Bilder als Medien, sondern zum Beispiel auch Videos. YouTube- und Vimeo-Videos können ganz einfach durch die Eingabe ihres URL mit Standardbordmitteln von TYPO3 CMS in die Website eingebunden werden.

Bereits seit Version 7.2 haben Redakteure außerdem die Möglichkeit, bestimmte Ausschnitte von Bildern sehr komfortabel im Backend zu bearbeiten (Image Cropping). Diese Funktion erlaubt es, Bilder auf bestimmte Größen zuzuschneiden, ohne dabei das Originalbild zu verändern (Bild 3).

Diese Image Cropping-Funktionalität steht außerdem Systemintegratoren in TypoScript, Fluid und im TCA (Table Configuration Array) zur Verfügung.

Primär für Systemintegratoren und Administratoren interessant ist die neue System-Extension Fluidbased Content Elements, die erst in den jüngsten Sprint-Releases eingeführt wurde. Diese erlaubt es, Fluid Templates anstelle des bisher gewohnten TypoScript zum Rendern von Inhaltselementen zu verwenden. Zwar ist diese Lösung noch ziemlich neu, sie könnte aber über kurz oder lang eine leistungsstarke Alternative zu CSS Styled Content darstellen.

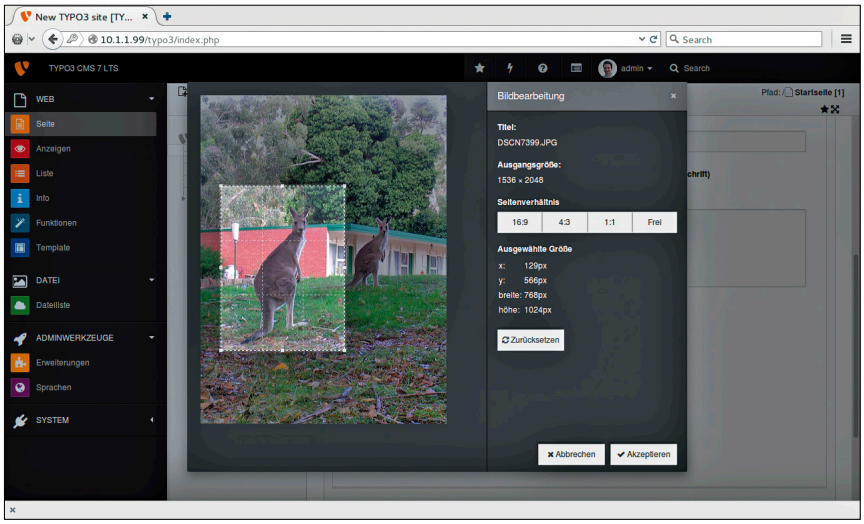
Backend-Benutzer

Um den sozialen Aspekt der Zusammenarbeit von Redakteuren zu stärken, haben Backend-Benutzer nun die Möglichkeit, in ihrem Konto ein Profilbild hochzuladen. Dieses kleine Bild, auch als Avatar bekannt, wird an verschie- ▶

denen Stellen im Backend dargestellt. Auch bei der Sicherheit hat sich bei den Backend-Benutzerkonten etwas verändert: Möchte ein Benutzer sein Passwort ändern, ist die vorherige Eingabe des alten Passworts zwingend notwendig.

Dateiliste

Drei neue Funktionen erweitern den Funktionsumfang der Dateiliste. Besonders bei großen Websites mit vielen Dateien und einer umfangreichen Verzeichnishierarchie war es bisher unter Umständen schwierig, Dateien im Filesystem zu finden. Durch die Eingabe eines beliebigen Suchbegriffs in einem Textfeld werden die Verzeichnisse rekursiv durchsucht und entsprechende Suchergebnisse aufgelistet.



Im Backend integrierte Bildbearbeitungsfunktion (Bild 4)

Die zweite interessante Neuerung betrifft das Hochladen von Dateien, sofern bereits eine Datei mit demselben Namen vorliegt. Hier wird nun ein Dialogfenster eingeblendet, in dem der Benutzer aufgefordert wird, zu wählen, welche der folgenden drei Aktionen ausgeführt werden soll: ersetzen, umbenennen oder überspringen. Dies kann für jede Datei einzeln sowie auch als Massen-Update angewendet werden (Bild 4). Um eine einzelne Datei ganz gezielt zu ersetzen, bietet die Dateiliste unter TYPO3 CMS jetzt einen extra Button in der Listenansicht. Eventuell existierende Metadaten gehen hierbei nicht verloren, sondern werden für die neue Datei automatisch übernommen.

Erweiterungs-Manager

Die neue TYPO3-CMS-Version bietet aber nicht nur Redakteuren viele Verbesserungen. Systemintegratoren können nun bei der Aktualisierung von Extensions die gewünschte Zielversion auswählen. Das ist besonders dann nützlich, wenn man nicht auf die neueste Version einer Extension aktualisieren möchte (dies konnte bisher nur über Umwege erreicht werden).

Branding bei der Anmeldung am Backend

So verleiht man dem Backend ein individuelles Outfit.

Ist man mit einem Administrator-Benutzer angemeldet, können im Erweiterungs-Manager die Einstellungen der Systemextension TYPO3 Backend aufgerufen werden. Hier kann nicht nur das TYPO3-Logo gegen ein beliebiges Bild ausgetauscht und eine individuelle Hervorhebungsfarbe ausgewählt, sondern auch ein eigenes Hintergrundbild hochgeladen werden. Letzteres wird dann bei Bildschirmgrößen von mindestens 767 Bildpunkten dargestellt – ideal ist dies für Agenturen, um der TYPO3-Instanz schon bei der Backend-Anmeldung ein Branding zu verleihen.

Der Erweiterungs-Manager kann jetzt außerdem so eingestellt werden, dass Extensions nach dem Download nicht automatisch installiert werden.

TypoScript

Zu den wichtigsten Änderungen im Bereich TSconfig und TypoScript gehört unter anderem die Tatsache, dass Backend-Layouts sich nun per PageTSconfig definieren lassen und somit die Konfiguration in Dateien ausgelagert werden kann. Web-Developer, die für die Entwicklung von TYPO3-Websites eine Versionsverwaltung wie zum Beispiel Git verwenden, werden diesen Schritt zweifelsfrei begrüßen.

Der TYPO3-interne Rich Text Editor (RTE) wurde an mehreren Stellen aktualisiert. Dazu gehört die Möglichkeit, nun auch mehrere CSS-Dateien einbinden zu können, um zum Beispiel eigene Styles zu definieren:

```
RTE.default.contentCSS {
  file1 = fileadmin/rte_stylesheet1.css
  file2 = fileadmin/rte_stylesheet2.css
}
```

Mit sogenannten Conditions kann man in TypoScript Konfigurationen vornehmen, die nur unter bestimmten Umständen gültig sein sollen. Die von TYPO3 bereitgestellten Conditions reichen hierbei von der Abfrage der aktuellen IP-Adresse des Besuchers über den Anmeldestatus bis hin zu Parametern im *GET/POST*-Request.

Neu eingeführt wurde in TYPO3 CMS 7 LTS ein API, das Custom Conditions zulässt und es somit Entwicklern erlaubt, ihre eigenen Conditions für TypoScript zu implementieren, ohne eine *userFunc* verwenden zu müssen. Ein Code-Beispiel findet sich in der offiziellen TypoScript-Referenz auf https://docs.typo3.org.

Zwei Erweiterungen betreffen das Einbinden von JavaScript-Dateien. Zum einen kann mit der neue Eigenschaft *integrity* ein SRI Hash (SubResource Integrity) spezifiziert werden. Bei SRI handelt es sich um eine W3C-Spezifikation, mit

der sichergestellt werden kann, dass Dateien, die sich auf Servern von Drittanbietern befinden (zum Beispiel jQuery von einem CDN), nicht manipuliert wurden.

Zum anderen wurde die Eigenschaft *async* eingeführt, mit der man das asynchrone Laden von JavaScript-Dateien konfiguriert. Ein Update auf jQuery UI Version 1.11 verspricht beschleunigte Ladezeiten im Backend, da ab dieser Version AMD (Asynchronous Module Definition) unterstützt wird. Dadurch werden JavaScript-Dateien erst geladen, wenn sie benötigt werden.

Neues für Entwickler

Es ist bekannt, dass bei dem TYPO3-CMS-Projekt großer Wert auf hohe Code-Qualität, einen guten Programmierstil und professionelle Programmierkonzepte gelegt wird. Daher wurden in den vergangenen Monaten auch die Unit- und Functional-Tests erweitert und alte Teile der PHP-Codebasis überarbeitet. Die Entwickler stellten somit sicher, dass TYPO3 CMS 7 LTS weitgehend den offiziellen Coding-Guidelines entspricht und der Kern einheitliche Standards einhält.

TYPO3 CMS Version 7.4 war übrigens das erste Open-Source-CMS, das den neuen PSR7-Standard implementiert hat. PSR (PHP Specification Request) ist eine Spezifikation, die der Standardisierung von Programmierkonzepten dient. Eine detaillierte Beschreibung der Spezifikation findet sich auf der Internetseite der PHP Framework Interop Group (www.phpfig.org).

Composer

Eine weitere wichtige Neuerung ist der Einsatz von Composer. Dieses mächtige Werkzeug ist aus der PHP-Welt nicht mehr wegzudenken, und auch das TYPO3-Projekt hat sich in den vergangenen Monaten verstärkt um dessen Unterstützung gekümmert. TYPO3 CMS kann nicht nur mit allen abhängigen Packages mittels Composer von Grund auf installiert werden, sondern unter http://composer.typo3.org steht ein zentrales Package Repository (Packagist) bereit, das sämtliche TYPO3-Extensions beinhaltet (Bild 5).

Extension-Entwickler können Package-Metadata und Installationsanweisungen in einer *composer.json*-Datei definieren. Der ComposerClassLoader in TYPO3 CMS berücksichtigt dies und unterstützt den PSR4 Class Standard (Improved Autoloading).

Form-Extension

Nicht zu verwechseln mit der zuvor erwähnten Form-Engine ist die sogenannte Form-Extension. Mit ihr können jegliche Arten von Formularen im Frontend entwickelt werden, ohne eine einzige Zeile PHP-Code schreiben zu müssen. Das können beispielsweise Kontaktformulare sein, die beim Absenden E-Mails an bestimmte Adressen senden, Benutzerregistrierungen und so fort. Die Form-Extension wurde von Grund auf überarbeitet und nutzt jetzt das Extbase-Framework und Fluid zum Rendern von Templates. Letzteres macht es Systemintegratoren besonders einfach, eigene Templates zu verwenden und somit Formulare individuell zu gestalten.

Funktionen in der Kommandozeile

Wenn man die zuvor beschriebene ComposerFunktionalitäten außer Acht lässt, hat sich für Systemadministratoren nicht sehr viel geändert. Erwähnenswert ist allerdings die Tatsache, dass die Console-Komponente aus dem Symfony-Projekt in den CommandController von TYPO3 integriert wurde. Diese bietet einige interessante Features, und Systemadministratoren werden mit hoher Wahrscheinlichkeit zukünftig mehr TYPO3 Funktionen über die Kommandozeile steuern können.

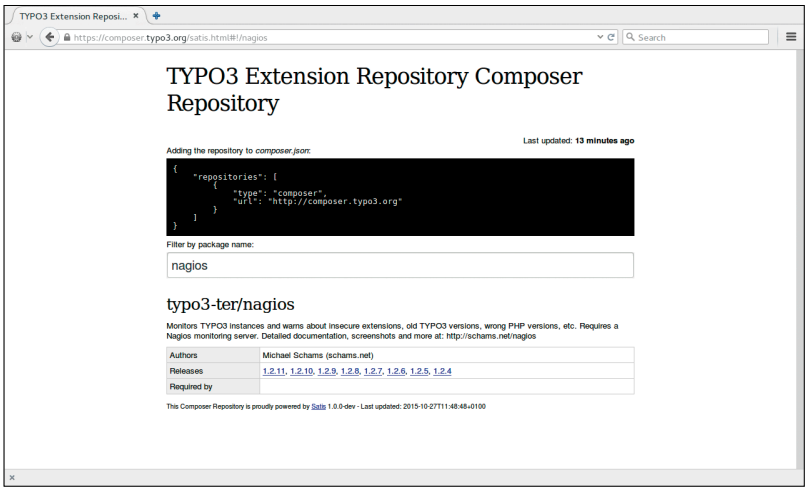
Eine weitere Neuerung ist der Parameters *s*, mit dem ein laufender Task im Scheduler über die Kommandozeile beenden werden kann.

Entfernte Funktionen und Kompatibilität

Die Veröffentlichung von sogenannten Major Releases ist immer ein guter Zeitpunkt, sich von Altlasten zu trennen. Die TYPO3-Entwickler halten sich hierbei strikt an die offizielle Deprecation Strategy, die regelt, dass eine Funktion zuerst als veraltet markiert werden muss und erst zwei TYPO3-Versionen später entfernt werden darf. Somit haben Systemintegratoren und Entwickler ausreichend Zeit, entsprechende Maßnahmen zu ergreifen.

Mit der Veröffentlichung von TYPO3 CMS 7 LTS ist wie erwartet eine große Anzahl veralteter Funktionen und Optionen entfernt worden. Dazu zählen zum Beispiel die TypoScript-Templates der Extension CSS Styled Content der TYPO3-CMS-Versionen 4.5 bis 6.1. Diese wurden lange Zeit aus Kompatibilitätsgründen in jeder neuen Version übernommen, nun aber entfernt.

TypoScript Conditions wurden bereits zuvor erwähnt. Die TYPO3-eigenen Conditions *browser*, *version*, *system* und *useragent* sind bereits seit TYPO3 CMS 7.0 nicht mehr vorhanden. Zudem wurde in derselben Version der Kompatibilitäts-Layer entfernt. Dieser hat in TYPO3 Version 6.2 noch ▶



TER Composer Repository für Extensions (Bild 5)

dafür gesorgt, dass Extensions, die keine PHP-Namespaces verwenden, lauffähig bleiben. Diese Rückwärtskompatibilität hat allerdings ihren Preis in Form von nicht unerheblichen Performance-Einbußen.

Wer in TYPO3 CMS 7 LTS nicht auf den besagten Kompatibilitäts-Layer verzichten kann, findet ihn in der Systemextension *compatibility6* wieder, die mitgeliefert wird und bei Bedarf einfach aktiviert werden kann. In diese Extension wurden übrigens noch weitere veraltete Features verschoben, die zuvor im TYPO3-Kern vorhanden waren. Dazu gehören etwa die TypoScript-cObjects *SEARCHRESULTS, COLUMNS, OTABLE, CLEARGIF, IMGTEXT, CTABLE, HRULER* sowie die Inhaltselemente *mailform* und *search*.

Durch den Fokus auf Twitter Bootstrap, jQuery und RequireJS konnten die JavaScript-Bibliotheken prototype, script.aculo.us und ExtCore entfernt werden. Dasselbe gilt für Teile von ExtJS, zum Beispiel die Flash- und Chart-Komponenten. Somit verlieren die TypoScript-Eigenschaften *page.javascriptLibs.Prototype, *.Scriptaculous, *.ExtCore* und *.ExtJs* ebenfalls ihre Gültigkeit.

Installation und Update

Abgesehen von der zuvor erwähnten Möglichkeit, TYPO3 CMS nun auch über Composer installieren zu können, funktioniert das bisher bekannte Vorgehen für eine Neuinstallation natürlich auch weiterhin: Nach dem Entpacken des Quellpakets werden die symbolischen Links zu den entsprechenden Verzeichnissen und Dateien angelegt, die Datei *FIRST_INSTALL* erstellt, und schon steht der webbasierten Installation von TYPO3 CMS 7 LTS nichts mehr im Weg.

Um eine vorhandene TYPO3-Installation auf die neue Version zu aktualisieren, muss diese in der Version 6.2 LTS vorliegen. Ältere Versionen sollten zuerst auf Version 6.2 gebracht werden. Abhängig von den verwendeten Extensions verläuft das Update problemlos, und der im Install-Tool integrierte Upgrade-Wizard unterstützt Administratoren bei diesem Prozess.

Mindestvoraussetzungen

Vor einer Neuinstallation und natürlich vor einem Update sollte auf jeden Fall sichergestellt werden, dass der Server die Mindestvoraussetzungen für die neue Version von TYPO3 CMS erfüllt. Als Middleware wird mindestens PHP Version 5.5 (besser 5.6) benötigt, und als Datenbankserver wird MySQL 5.5 oder 5.6 empfohlen. MySQL darf nicht im *strict mode* betrieben werden, was bei Version 5.6 allerdings häufig die Standardeinstellung ist.

Abgesehen davon empfehlen die TYPO3-Entwickler mindestens folgende serverseitigen Einstellungen: 128 MByte Speicher für PHP, die Einstellung von 240 Sekunden als maximale Ausführungszeit für PHP-Skripts *(max_execution_time)* und 8 MByte für Datei-Uploads.

Da das Backend von TYPO3 CMS unter Umständen einige sehr umfangreiche Formulare enthalten kann, ist es außerdem empfehlenswert, die PHP-Einstellung *max_input_vars* auf den Wert 1500 zu setzen (üblicherweise steht dieser Wert auf 1000).

Links zum Thema

- Projektseite
  http://typo3.org
- Dokumentation
  https://docs.typo3.org
- PHP Framework Interop Group
  www.phpfig.org
- Composer
  http://getcomposer.com

Fazit

Die vorherige TYPO3 CMS Version 6.2 LTS hatte ihren Schwerpunkt auf der Architektur des Systems (zum Beispiel die Einführung von Namespaces et cetera). Der TYPO3-CMS-Version 7 LTS ist deutlich anzumerken, dass nun wieder Endbenutzer – also zum Beispiel Redakteure – im Fokus stehen. Zusätzlich haben die Entwickler es geschafft, eine LTS-Version auf den Markt zu bringen, deren Kern technologisch zukunftsweisend ausgerichtet ist.

Die Ereignisse der vergangenen Monate haben bei vielen TYPO3-Anhängern für etwas Unsicherheit gesorgt: Die Veröffentlichung von Neos CMS Version 2.0 und die Abspaltung von der TYPO3 Association warfen erneut Fragen über die Zukunft von TYPO3 auf – einem CMS, das bereits seit über 14 Jahren auf dem Markt ist.

Die neue LTS Version zeigt allerdings ganz klar, dass TYPO3 CMS auch weiterhin eine der mächtigsten PHP-Applikationen ist, um Websites aller Art, Größe und Komplexität zu entwickeln. Das visuell und technisch überarbeitete Backend, die kinderleichte Integration von Videos, die neue Bildbearbeitungsfunktion sowie die neuen Features in der Dateiliste sprechen besonders Redakteure an.

Systemintegratoren werden von den Verbesserungen in den Bereichen TypoScript, Extension Manager und User Interface profitieren. Die modernen Technologien, um die die Codebasis erweitert wurde, und die gleichzeitige Entfernung von Altlasten haben den Kern wieder auf ein stabiles Fundament gehoben. Letzteres wird sich auch in der Performance positiv auswirken, vorausgesetzt, man kann auf alte Extensions verzichten. Es wird garantiert nicht viel Überzeugungsarbeit notwendig sein, wenn Agenturen ihren Kunden die neue Version von TYPO3 CMS als Lösung vorschlagen. ■

Michael Schams
lebt seit 2008 in Melbourne/Australien und arbeitet als Projekt Manager und Consultant. Er ist zertifizierter TYPO3 Integrator und Project Leader des offiziellen TYPO3 Security Guides.
@MichaelSchams

Internet World
Die E-Commerce Messe

01.-02. März 2016, München

Kongress-Programm online!

Die Zukunft des E-Commerce

Julia Bösch
Outfittery GmbH

Jens Diekmann
Douglas AG

Moritz Hau
Zalando SE

Prof. Dr. Gerrit Heinemann
HS Niederrhein

Dominik Hensel
Deutsche See GmbH

Moritz Keller
Keller Sports GmbH

Franziska Majer
Videdressing SAS

Nils Müller
TRENDONE GmbH

Dr. Marcus Schöberl
Amazon.de GmbH

Dr. Stephan Zoll
Ebay International AG

Mit Code IW16wmd 240,– € sparen!

internetworld-messe.de

InternetWorldMesse
#iwm

WEBSHOP-LÖSUNGEN MIT FLOW

Flow-Commerce

Mit dem Aimeos-Package lassen sich maßgeschneiderte Webshop-Lösungen bauen.

Seit einiger Zeit steht für das PHP-Framework Flow ein leistungsfähiges E-Commerce-Package zur Verfügung, mit dem sich auch anspruchsvolle Wünsche realisieren lassen. In diesem Workshop werden wir die Einbindung in eine Flow-Applikation beleuchten und uns einen Überblick über die mitgelieferte Funktionalität verschaffen.

Hintergründe

Das Flow-Package ist Teil des Open-Source-Projekts Aimeos, das auf Basis einer gemeinsamen E-Commerce-Bibliothek Webshop-Komponenten für verschiedene PHP-Frameworks und Applikationen anbietet. Bis jetzt werden Laravel, Flow, Symfony2, Zend und das TYPO3-CMS unterstützt.

Auf der Webseite http://aimeos.org findet man weitere Informationen zum Projekt, zur Community und den anderen Teilprojekten.

Die Grundlage dafür ist die gemeinsame PHP-E-Commerce-Bibliothek (http://github.com/aimeos/aimeos-core), die alle notwendigen Funktionalitäten eines Webshops als anpassbare Komponenten kapselt. Sie ist plattformunabhängig und benötigt nur PHP 5.3 sowie einige Module, die in der Regel in jeder PHP-Standardinstallation enthalten sind.

Die Integration in das jeweilige Framework oder die Applikation erfolgt durch Adapter, die eine einheitliche Schnittstelle zu den Objekten des Wirtssystems bereitstellen. So wird zum Beispiel die native Cache-, Logger- und Session-Implementierung von Flow genutzt, und URLs werden durch den Flow-UriBuilder erzeugt. Natürlich können auch alle Konfigurationsoptionen, mit denen sich die Bibliothek an eigene Bedürfnisse anpassen lässt, durch die Flow-YAML-Dateien geändert werden.

Die E-Commerce-Bibliothek ist auf den Umgang mit großen Produktmengen bei gleichzeitig geringen Responsezeiten optimiert. Bis zu 100.000 Produkte sind ohne zusätzliche Dienste wie Redis oder ElasticSearch mit einer MySQL-Datenbank problemlos möglich. Im lokalen Setup einer Laravel-Applikation wurden Response-Zeiten bis zu 40 Millisekunden gemessen, mit Flow liegen diese aber etwas höher.

Installation

Eine neue Flow-Applikation lässt sich am einfachsten mit Composer und der Flow-Distribution erstellen. Aktuell ist die Version 3.0, und Voraussetzung dafür ist die Installation von

Listing 1: Configuration/Settings.yaml

```
TYPO3:
  Flow:
    persistence:
      backendOptions:
        host: 'Hostname oder IP-Adresse'
        dbname: 'Datenbankname'
        user: 'Datenbanknutzer'
        password: 'geheimes Passwort'
```

Listing 2: composer.json

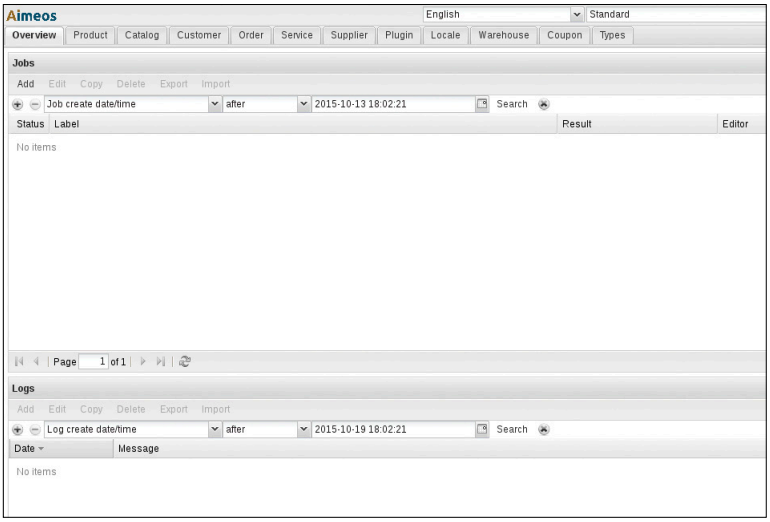
```
"require": {
  "aimeos/aimeos-flow": "~1.0",
  ...
},
"extra": {
  "installer-paths": {
    "Packages/Extensions/{$name}/":
    ["type:aimeos-extension"]
  }
},
```

Composer. Nähere Information dazu findet man auf http://getcomposer.org. Für eine neue Basisapplikation muss folgender Befehl auf der Kommandozeile ausgeführt werden:

```
composer create-project
typo3/flow-base-distribution tutorial
```

Danach steht eine Flow-Applikation zur Verfügung, die noch konfiguriert werden muss. Für Aimeos ist zumindest die Einrichtung einer Datenbank und das Hinterlegen der Zugangsdaten in der Flow-Datei *Configuration/Settings.yaml* erforderlich (Listing 1).

Bei den YAML-Dateien ist wichtig zu wissen, dass jede Einrückung eine Ebene in der Konfiguration darstellt und Einrückungen immer aus Vielfachen von zwei Leerzeichen bestehen. Andernfalls wird Flow eine Fehlermeldung wegen eines ungültigen Konfigurationsformats ausgeben.



Admin: Die auf ExtJS basierende Administrationsoberfläche (Bild 2)

Danach kann das Aimeos-Package installiert werden. Da Aimeos ebenfalls Composer benutzt, reicht es, die Zeilen wie in Listing 2 in die bereits vorhandene *composer.json*-Datei einzupflegen. Im Abschnitt *require* sind bereits einige Abhängigkeiten aufgelistet. Hier muss nur diejenige für das Aimeos-Package hinzugefügt werden. Der *extra*-Abschnitt wird für die Aimeos-Erweiterungen benötigt, zu denen auch die Adapter für die Integration in das Flow-Framework gehören. Anschließend reicht ein

```
composer update
```

im Hauptverzeichnis der Flow-Applikation aus, um alle notwendigen Abhängigkeiten zu installieren.

Setup

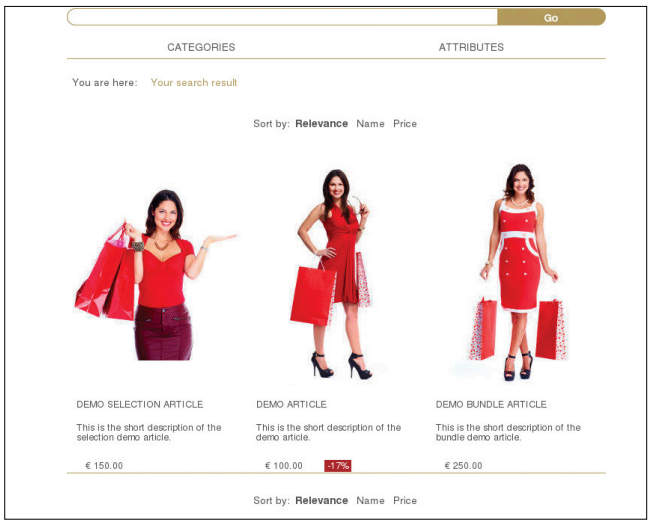
Noch sind die Aimeos-Komponenten nicht lauffähig, denn dazu fehlen noch die notwendigen Tabellen in der Datenbank. Derzeit unterstützt Aimeos nur MySQL ab 5.1, für das es auch optimiert ist. Die Datenbankstruktur lässt sich automatisch mit folgendem Flow-Befehl auf der Kommandozeile anlegen:

```
./flow aimeos:setup --option=setup/default/demo:1
```

Dadurch werden nicht nur die zu Aimeos gehörenden Tabellen und Indizes angelegt, sondern auch ein kompletter Demo-Datensatz eingespielt. Es entsteht damit ein vollständig lauffähiger Webshop, mit dem Sie sofort loslegen können. In Produktionsumgebungen würde man den Befehl ohne *--option=...* ausführen, das auch vorhandene Demodaten wieder entfernt, ohne selbst angelegte Datensätze zu löschen. Bringen neue Versionen Änderungen an der Datenbankstruktur mit sich, wird ebenfalls das Schema automatisch aktualisiert und alle vorhandenen Daten migriert.

Als letzter Schritt sollten für SEO-freundliche URLs noch die vom Aimeos-Package mitgelieferten Routen importiert werden. Dazu müssen die Zeilen wie in Listing 3 gezeigt am Anfang der *Configuration/Routes.yaml* eingefügt werden.

Auch hier ist wieder auf die korrekte Einrückung der Zeilen zu achten. Wichtig ist außerdem, dass die Zeilen vor der Definition der *FlowSubRoutes* eingefügt werden. Andernfalls passen die Flow-Standardrouten zunächst und es erscheint eine Fehlermeldung, dass die Route für das Aimeos-Package nicht gefunden wurde. Zum schnellen Ausprobieren der ▶



Frontend: Listenseite mit dem Aimeos-Standard-Theme (Bild 1)

Listing 3: Configuration/Routes.yaml

```
-
  name: 'Aimeos'
  uriPattern: 'shop/<AimeosShopRoutes>'
  subRoutes:
    AimeosShopRoutes:
      package: 'Aimeos.Shop'
```

Applikation in einer lokalen Installation eignet sich der ab PHP 5.5 integrierte Webserver sehr gut. Er wird auf der Kommandozeile mit dem folgenden Befehl im Hauptverzeichnis der Applikation gestartet:

```
php -S 127.0.0.1:8000 -t Web
```

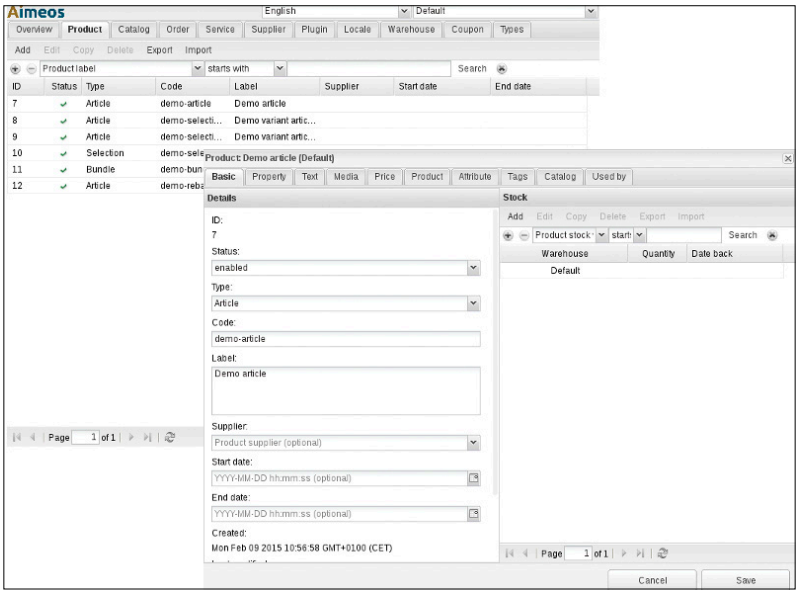
Anschließend stehen das Frontend (Bild 1) und die Administrationsoberfläche (Bild 2) von Aimeos unter den URLs *http://127.0.0.1:8000/shop/list* und *http://127.0.0.1:8000/shop/admin* zur Verfügung.

Komponenten

In der Listenansicht sieht man bereits unterschiedliche Komponenten, die auf dieser Seite plaziert wurden (Bild 3). Dazu gehört neben der Produktliste an sich auch der Filter für die Kategorien und die facettierte Suche sowie der kleine Warenkorb im oberen Bereich. Insgesamt sind bereits folgende Seiten vorkonfiguriert:

- Listenansicht,
- Detailansicht,
- Warenkorb,
- Checkout-Prozess,
- Bestätigungsseite,
- MyAccount-Seite.

Darüber hinaus gibt es noch Hilfskomponenten für die facettierte Suche, die Volltextsuche oder für die Aktualisierung des Versand- und Zahlungsstatus. Diese sind unter eigenen URLs ansprechbar und geben Daten im JSON-Format, als JavaScript oder nach spezifischen Anforderungen zurück. Insgesamt gibt es derzeit 19 verschiedene Komponenten (inklusive diejenigen für den Versand der verschiedenen E-Mails), die jeweils eine spezifische Aufgabe erfüllen. Die meisten auf den genannten Seiten eingebundenen Komponenten bestehen selbst aus zahlreichen Teilkomponenten, die zusammen eine hierarchische Struktur bilden. Sowohl die Kompontenen als auch deren Teilkomponenten lassen sich durch Konfiguration beliebig auf den Seiten beziehungsweise innerhalb einer Komponente verschieben. Dadurch ist es möglich, die resultierende HTML-Struktur alleine durch die Anpassung der Konfiguration zu ändern.



Komponenten, die auf der Demo-Seite plaziert sind (Bild 3)

Genauso können vorhandene Teilkomponenten entfernt, neue hinzugefügt oder bestehende durch eigene ersetzt werden. Falls das strukturelle HTML einmal nicht alleine durch Konfiguration und CSS an die Bedürfnisse des Kunden angepasst werden kann, lässt sich natürlich jedes Template einer Teilkomponente austauschen. Dadurch wird der Aufwand für Anpassungen weitestgehend minimiert. Mehr dazu ist in der Dokumentation auf http://aimeos.org/docs/Flow zu finden.

Fazit

Mit dem Aimeos Package kann innerhalb kurzer Zeit jede Flow-Applikation um einen Webshop oder eine flexible E-Commerce-Lösung ergänzt werden. Mit circa 170.000 Zeilen Code, über 10.000 Tests, mehr als 100.000 Zeilen Dokumentation sowie circa 2500 Konfigurationsoptionen bietet Aimeos ein leistungsfähiges und gut getestetes System, das auch hohen Ansprüchen problemlos genügt. Die Einbindung in Neos als CMS wäre der nächste logische Schritt. ■

Links zum Thema

- Open-Source-Projekt Aimeos
  http://aimeos.org
- Aimeos Flow Package
  http://github.com/aimeos/aimeos-flow
- Aimeos E-Commerce Core Bibliothek
  http://github.com/aimeos/aimeos-core
- Composer
  http://getcomposer.org
- PHP-Framework Flow
  http://flow.typo3.org
- Aimeos-Dokumentation
  http://aimeos.org/docs/Flow

Norbert Sendetzky
ist Software-Architekt und Berater mit den Schwerpunkten E-Commerce, Performance und Sicherheit. Des Öfteren findet man ihn auf Konferenzen und Barcamps, wo er auch gerne in Sessions darüber erzählt.

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

GRAFIK FÜR ENTWICKLER

Gut gelandet

Text- und Bildaufbau bestimmen die Qualität einer Landingpage.

Bei einer Landingpage handelt es sich um die Seite, die sich nach einem Klick auf einen Link in einer Suchmaschine oder auf ein Banner öffnet. Oft ist es die Startseite oder eine spezielle Unterseite, die für den Besucher immer ein einziges bestimmtes Angebot oder Produkt in den Vordergrund stellt und für bestimmte Keywords optimiert wurde.

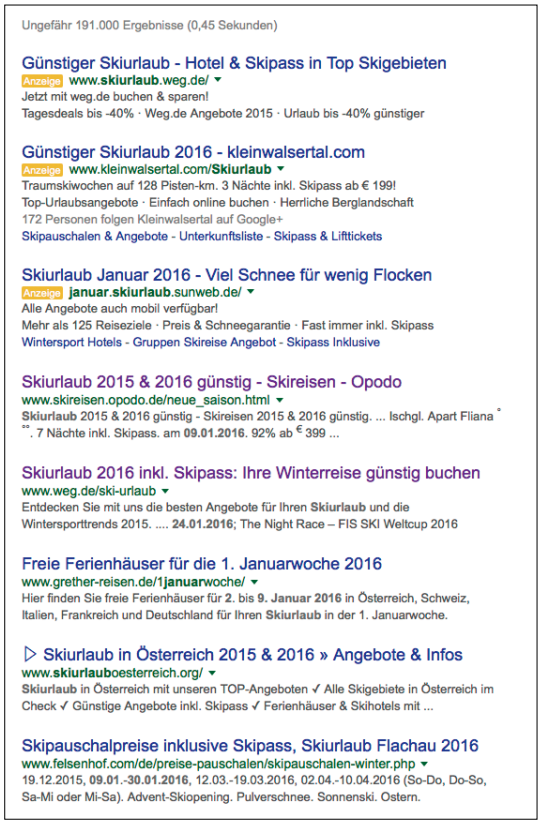
Gibt der Suchende etwa „Skiurlaub Januar 2016“ bei Google oder einer anderen Suchmaschine ein, bekommt er eine Liste von Ergebnissen: Zuoberst stehen die Anzeigen, die etwa über Google Adwords erstellt worden sind. Unter diesen Anzeigen erscheinen die Links zu weiteren Seiten, die die Suchmaschine anhand der Keywords findet. Im Beispiel sind das fast 200.000 Ergebnisse (Bild 1).

In der Regel klickt der Besucher auf einen Link, der sich auf der ersten Seite befindet – wenn die kurzen Beschreibungstexte zum Gesuchten passen. Im Beispiel bietet die Seite www.weg.de/ski-urlaub eine Headline, die besonders vielversprechend klingt: »Skiurlaub 2016 inkl. Skipass: Ihre Winterreise günstig buchen«. Jetzt kommt es darauf an, dass genau diese Seite, die damit zur Landingpage wird, die gewünschten Informationen liefert (Bild 2).

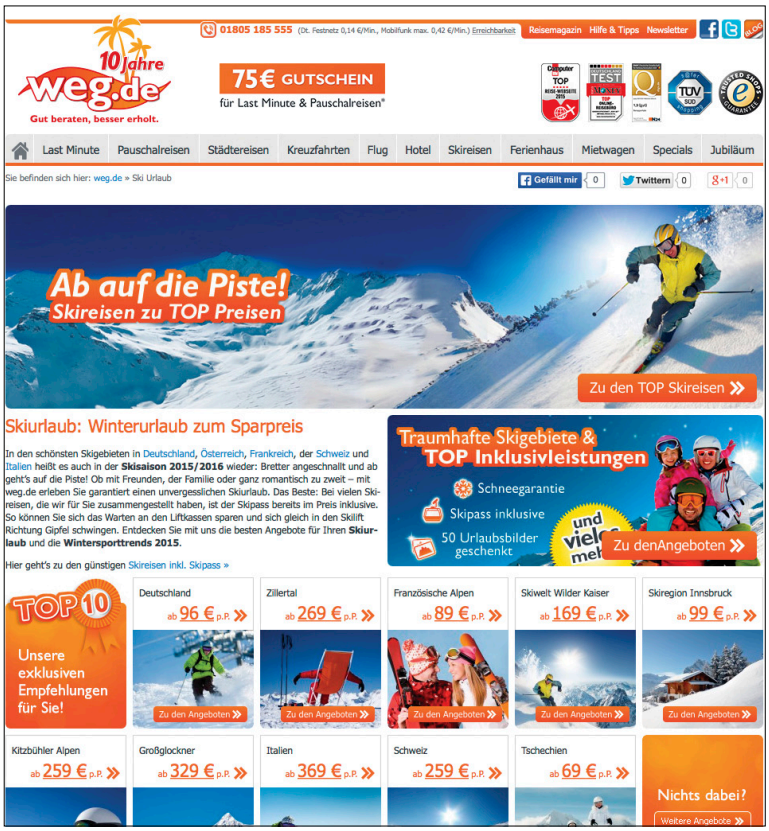
Landet der Besucher auf einer Seite, wird er in nur wenigen Sekunden intuitiv entscheiden, ob die Seite das Versprochene bietet, und vor allem, ob sie vertrauenswürdig ist oder nicht. Etwa zwanzig bis dreißig Sekunden verweilen die meisten Besucher auf einer gerade geöffneten Seite. Manche unterschreiten diesen Wert sogar, geduldige User lassen sich bis zu 40 Sekunden auf den gezeigten Webinhalt ein.

Wichtig ist also eine klar erkennbare Struktur. Die Seiteninhalte, ob Text, Grafik, Bilder oder Videos, müssen eindeutige Informationen zum Produkt enthalten, schnell erfassbar und auf die entsprechende Zielgruppe zugeschnitten sein. Steht das Produkt direkt zum Verkauf, sollten zudem schlüssige Zahlungsmethoden angeboten werden, um dem Interessenten die Entscheidung zum Kauf zu erleichtern (Bild 3).

Zudem muss die technische Seite stimmen: Zu lange Ladezeiten der Inhalte verschlingen einen Löwenanteil der Verweildauer des Besuchers.

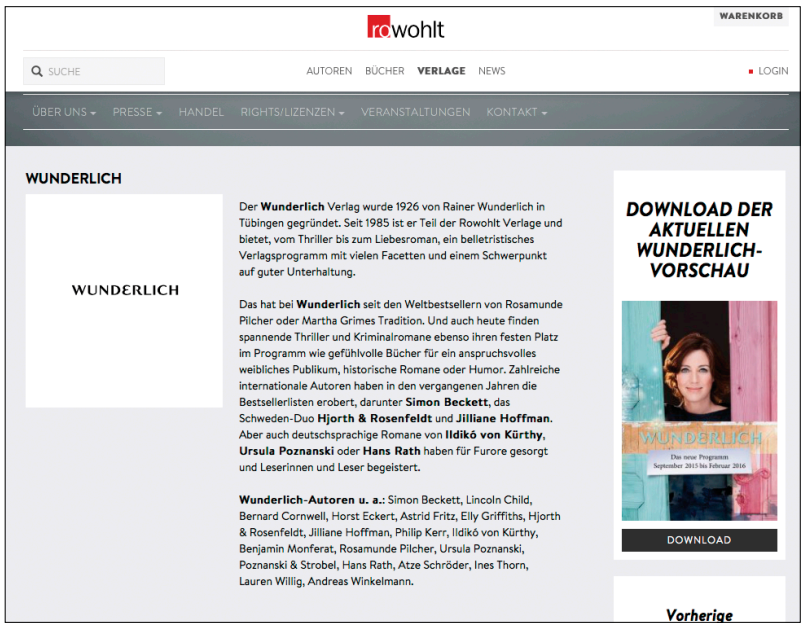


Die Suche bei Google nach »Skiurlaub Januar 2016« liefert fast 200.000 Ergebnisse (Bild 1)



Weg.de zeigt übersichtlich alle Elemente, die auf eine Landingpage gehören (Bild 2)

Die Landingpage von Windows 10 ist dank der klaren Struktur schnell zu erfassen (Bild 3)



Die einzelnen Verlage der Gruppe Rowohlt werden auf eigenen Landingpages in einem eigenen Text vorgestellt (Bild 4)

Eine Landingpage kann die übergeordnete Seite einer Werbekampagne sein und so dem Besucher einen ersten Überblick über die Angebote und den Anbieter liefern. Doch auch andere Einsatzmöglichkeiten sind denkbar: Geht es beispielsweise darum, sich als Verlag auf dem Markt zu platzieren, sollten auch hier gut funktionierende Einstiegsseiten den Betrachter zum Landen einladen. Doch welche Elemente machen hier eine gute Landingpage aus? Hier reicht es nicht, beispielsweise nur einzelne Bücher auf die Seite zu packen, vielmehr gehört auf die Landingpage eigener Inhalt (Unique Content). Schließlich will der Interessent zunächst einen groben Überblick bekommen, um danach gezielt auf Unterseiten weitere Inhalte zu betrachten. Zudem ist Unique Content eine der wichtigen Voraussetzungen für eine gute Positionierung auf der Liste der Suchergebnisse: Suchmaschinen durchpflügen lediglich den Text.

Beispiel Rowohlt: Jeder Ast der Verlagsgruppe hat eine eigene Landingpage, auf der ein übergeordneter Text den Verlag und seine Entstehung und Ausrichtung beschreibt und die wichtigsten Autoren nennt (Bild 4).

Auf Zielgruppe orientiert

Doch nicht nur der Text zählt. Eine Landingpage konzentriert sich auf ein bestimmtes Angebot und präsentiert dabei dem Betrachter die Inhalte so, dass sie auf seine Bedürfnisse angepasst sind. Gibt es viel Konkurrenz, bestimmt der erste Eindruck des Betrachters, welches Angebot er sich näher ansieht – und ob er eventuell zum Kunden wird. Dieser erste Eindruck hängt jedoch nicht nur von den Texten ab.

Ein wichtiger Beurteilungsfaktor des Users ist, ob er die Seite sympathisch findet. Und Bilder sind Sympathieträger. Zudem müssen die Bedürfnisse der angepeilten Zielgruppe berücksichtigt werden. Wird etwa nach betreutem Wohnen für Senioren gesucht, darf die Seite nicht überladen sein.

Ältere Menschen sind mit dem Internet noch nicht so vertraut und kommen mit einer Seite besser zurecht, die nur wenige Elemente zeigt. Neben einer größeren Schrift oder der Möglichkeit, diese zu vergrößern, dürfen Links zu wichtigen Informationen nicht fehlen (Bild 5).

Bei der Konzeption der Landingpage sollte immer das angebotene Produkt der Seite im Zentrum liegen. Seitenverweise, die mit dem eigentlichen Angebot nichts zu tun haben, haben hier also in der Regel nichts zu suchen. Links zu den entsprechenden Unterseiten, etwa einzelnen Produkten oder Ergänzungen, dürfen mit auf die Seite. ▶

Diese Seite spricht die Zielgruppe Senioren an: Wichtig sind stimmungsvolle Bilder, wenige Elemente auf der Seite und gut lesbare Schrift (Bild 5)

Ein Beispiel: Das Delikatessengeschäft Dallmayr bietet neben verschiedenen Produkten auch Serviceleistungen wie Party und Catering. Die Seite für den hauseigenen Kaffee entspricht dem Aufbau einer guten Landingpage: Links oben liegt das Logo. Unter der nachfolgenden, dezent gestalteten Hauptnavigationsleiste liegt ein Bild als Stimmungsträger. Darunter ist die Headline zu sehen, der ein allgemeiner Text über Kaffee folgt. Rechts daneben befindet sich als Produktabbildung eine Kaffeebohne. Weitere, dazugehörige Links finden sich links neben dem Text. Durch die farbliche Gestaltung, weißer Text auf schwarzem Grund, heben sich die Seitenverweise gut von den restlichen Design-Elementen ab.

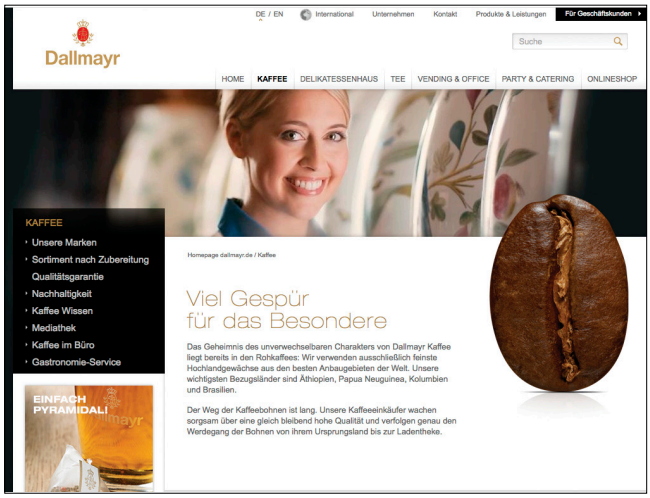
Da für den Verkauf ein Online-Shop über die obere Navigationsleiste erreichbar ist, kommt diese Landingpage ohne Call-to-Action oder Formular aus. Auch ohne den Text über Kaffee zu lesen, ist der Inhalt der Seite durch den klaren Aufbau in Sekundenschnelle zu erfassen. Zwar greift die die Headline nicht das Thema Kaffee auf, aber zusammen mit dem Produktbild, der Bohne und dem bekannten Dallmayr-Logo ist der Inhalt des Angebots dennoch auf den ersten Blick zu erkennen (Bild 6).

Call-to-Action

Bei einem Call-to-Action handelt es sich um ein Marketing-Element, das aus einer grafisch gestalteten Schaltfläche und einem kurzen dazugehörigen Erklärungstext besteht. Er dient dazu, den Besucher der Seite per Klick auf die Schaltfläche zu einer Handlung aufzufordern.

Bei der Gestaltung dieses Elements muss darauf geachtet werden, dass es seriös erscheint: Viele Effekte und Bildelemente sind hier fehl am Platz. Zudem hilft die passende Formulierung des Begleittextes, den Besucher zur Handlung zu animieren. Gibt es auf einer Landingpage einen Call-to-Action, sollte dieser möglichst so platziert werden, dass er ohne lästiges Scrollen erreicht werden kann (Bild 7).

Neben einem Call-to-Action kann eine Landingpage ein Formular enthalten. Dabei gilt es zu beachten: Je kürzer ein Formular ist, desto eher wird ein potenzieller Kunde es ausfüllen. Zudem sollte das Formular möglichst benutzerfreundlich sein: Scheint die Eingabe zu kompliziert zu sein, brechen viele Nutzer an dieser Stelle ab.



Die Kaffee-Seite von Dallmayr liefert einen klaren, gut strukturierten Aufbau (Bild 6)

Call-to-Action: Der Link zur kostenlosen Testversion einer Vereins-Software steht an oberen Ende der Seite (Bild 7)

Die Benutzerfreundlichkeit eines Formulars wird auch durch die entsprechende grafische Aufbereitung unterstützt. So sollte auf Hintergrundmuster auf der Schaltfläche oder auf aufdringliche Schlagschatten und 3D-Effekte verzichtet werden. Die Bezeichnung der Eingabefelder sollte ebenfalls wohlüberlegt sein: Am schlüssigsten sind die Einträge linksbündig über dem entsprechenden Feld. Auch ein hellgrauer Eintrag mitten im Feld ist benutzerfreundlich (Bild 8).

Zudem müssen die Größen und die Anordnung der Eingabefelder stimmen. Zu vermeiden sind beispielsweise zu kurze Felder, etwa für den Namen. Pflichtfelder müssen als solche gekennzeichnet sein.

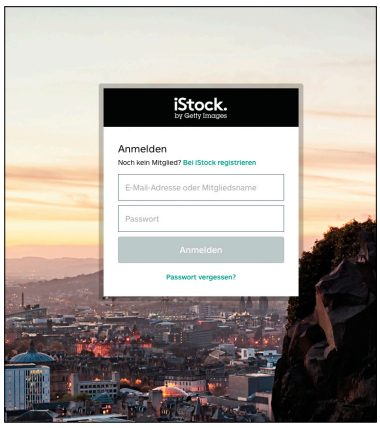
Erst wenn der potenzielle Käufer dem Anbieter ein gewisses Vertrauen entgegenbringt, wird er das Geschäft abschließen. Das gilt für einen Verkauf über die Ladentheke genauso wie beim Kauf über einen Online-Shop.

Ähnlich verhält es sich auch mit anderen Angeboten: Selbst wenn die Mitgliedschaft bei einem Verein kostenlos ist, werden die Umworbenen zurückschrecken, wenn dessen Ansinnen nicht klar erkennbar ist. Wichtig ist also Transparenz. Je offener das Angebot dargelegt wird, desto vertrauenswürdiger erscheint es. Statistiken, Zahlen oder andere belegbare Fakten schaffen Transparenz. Weitaus stärkere Trust-Faktoren stellen positive Testberichte oder Gütesiegel dar (Bild 9).

Strategien des Online-Marketings

Nicht nur die thematisch passenden grafischen Elemente und Texte sowie deren Anordnung sorgen für eine gut funktionierende Landingpage. Um eine Seite erfolgreich zu platzieren, sollte man die wichtigsten Strategien des Online-Marketings berücksichtigen: Eine umfassende Suchmaschinenoptimierung (SEO) erhöht das Ranking. Weiter erhöhen die sozialen Netzwerke wie Twitter, Facebook und Co den Bekanntheitsgrad und ermöglichen weitere Marketingmaßnahmen, sofern die Besucher der Seite folgen. Auch diese Links können dem herrschenden Seiten-Layout angepasst werden, um ein harmonisches Gesamtbild zu erzeugen.

Dazu kommt die Kontaktpflege zu den Kunden über E-Mail. Hier sorgen beispielsweise Newsletter dafür, den Kunden weitere Informationen zukommen zu lassen – und dadurch den Umsatz zu steigern. Neben Banner-Werbung er-



Das Formular zur Anmeldung bei iStock von Getty Images ist mit den beiden grafisch einfach gestalteten Eingabefeldern sehr benutzerfreundlich (Bild 8)

Gleich zwei Bio-Siegel auf einer Seite von Aldi-Süd erzeugen Vertrauen beim Verbraucher (Bild 9)

möglicht das Affiliate Marketing Kooperationen zwischen Anbietern und dem Seiteninhaber: Dabei werden gegen eine Vermittlungsprovision Werbe-Links platziert.

Das Sieben-Ebenen-Modell

Damit der Benutzer die gewünschte Interaktion erfolgreich abschließt, beispielsweise einen Verkauf, ist es sinnvoll, den Aufbau der Seite aus der Perspektive des Users zu planen. Diese Vorgehensweise wird Conversion-Optimierung genannt. Für die Planung eines derartigen Aufbaus gibt es das sogenannte Sieben-Ebenen-Modell.

Die erste Ebene, **Relevanz:** Entspricht der Inhalt dem, was gesucht wird? Um dem User wirklich das zu bieten, was er sucht, müssen die Schlüsselwörter so angelegt sein, dass der Suchende tatsächlich auf der richtigen Seite landet. Besonders bei einer exakten Eingabe der Suchbegriffe sollte das funktionieren.

Die zweite Ebene, **Vertrauen:** Ist die Seite vertrauenswürdig? Neben den bereits beschriebenen Trust-Faktoren schaffen auch grammatikalisch wie stilistisch einwandfreie Texte sowie ein professionelles Layout Vertrauen. Dazu gehören zum Beispiel auch individuelle Fotos; Bilder aus Datenbanken gilt es zu vermeiden. Auch hier kommt es wieder auf den ersten Eindruck an.

Die dritte Ebene, **Orientierung:** Ist das Gesuchte zu finden? Hier spielt, wie bereits erwähnt, ein klarer Seitenaufbau die entscheidende Rolle. Dazu sollten neben den schlüssigen Texten auch die Bilder ihr Übriges tun: Wurde beispielsweise nach Jacken gesucht, ist es zwar eine schöne Idee, die passenden Hosen daneben abzubilden, nach dem Gesetz der Nähe funktioniert diese Vorgehensweise jedoch nicht – sie ist für den Suchenden eher verwirrend. Zudem sollte die Navigation schlüssig und an einem präsenten Platz untergebracht sein.

Die vierte Ebene, **Stimulanz:** Ist der Anreiz groß genug? Hier geht es um Emotionen. Erst wenn diese beim Betrachter angesprochen werden, findet er die Befriedigung seines Begehrens auf der Seite. Wichtige Dinge gehören dabei in den Vordergrund – sei es durch die passenden Bilder, Kurztexte oder zur Not auch über Signalfarben. Auch eine begrenzte Verfügbarkeit kann zum Kauf verführen, Beispiel Amazon: Hier ist eine Systemkamera im Angebot, der günstige Preis reizt bereits zum Kauf. Zudem sind nur noch vier Stück auf Lager, eine Tatsache, die die Verführung noch verstärkt.

Die fünfte Ebene, **Sicherheit:** Ist der Abschluss riskant? Gibt es auf das Gekaufte eine Garantie, so muss die Landingpage das auch vermitteln. Rückgaberechte sollten ebenso deutlich kommuniziert werden wie eventuell anfallende Kosten für den Versand. Bei Online-Bestellungen, bei denen der Käufer ein Formular ausfüllen muss, sollte darauf hingewiesen werden, wie mit den persönlichen Daten umgegangen wird. Für einen Abschluss braucht der Betrachter das Gefühl, alles unter Kontrolle zu haben.

Die sechste Ebene, **Komfort:** Ist die Methode zum Zahlen einfach? Hier geht es darum, dem Nutzer den Abschluss nach einer positiven Entscheidung so leicht wie möglich zu machen. Dabei sollten Formulare nach dem bereits geschilderten Muster konzipiert sein. Sind mehrere Daten fällig, helfen Klappmenüs dabei, das Formular übersichtlich zu gestalten

Die siebte Ebene, **Bewertung:** Was kommt jetzt? Hier geht es um die nicht mehr um die eigentliche Gestaltung der Seite, sondern um alle Dinge, die nach dem Abschluss passieren. So fühlt sich der Käufer beispielsweise gut betreut, wenn er die Sendung nachverfolgen kann.

Fazit

Der Stellenwert einer guten Landingpage wird häufig unterschätzt – so wie auch der Aufwand, sie zu planen und zu finalisieren. Dieser Aufwand ist jedoch notwendig, um einen Internetauftritt erfolgreich zu platzieren. Die Konkurrenz im Netz wird immer größer – und im Internet ist es wie im wirklichen Leben: Liegt das Geschäft in einer kleinen Seitenstraße, so gibt es kaum Laufkundschaft. ■

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.

WISSENSBASIERTE APPLIKATIONEN ENTWICKELN MIT ONTOLOGIEN

Smarte Apps entwickeln

Semantische Web-Datenbanken mit OWL2 – eine Einführung.

Die Web Ontology Language (OWL) erfreut sich immer größerer Beliebtheit – für die semantische Suche im Web, für Informationssysteme und Wissensdatenbanken ebenso wie für smarte Softwaresysteme. Intelligente Maschinen benötigen nicht nur Daten, sondern Wissen – generiert durch Big-Data-Analysen und Machine-Based Learning.

Dies erlaubt ihnen, autonom zu handeln und zu interagieren. Maschinen werden zu intelligenten Assistenten, die das Verhalten ihrer Anwender verstehen und sich daran anpassen können. Immer mehr Applikationen bauen daher auf Wissen. Dieser Artikel zeigt Ihnen, wie Sie OWL2 zum Aufbau von Wissensdatenbanken einsetzen.

Die erste Version von OWL wurde bereits im Jahr 2004 verabschiedet – im Wesentlichen basierend auf der Description Logic (DL), verwaltet in XML und angetreten mit dem Ziel, die maschinelle Auswertbarkeit des Resource Description Frameworks (RDF) zu verbessern. Die praktischen Erfahrungen zeigten jedoch schnell einen Mangel an Konstrukten, die für die Modellierung komplexer Domänen notwendig waren. Beispielsweise war die Anzahl von Datentypen eingeschränkt, und es konnten keine fremden Ontologien durch Importe eingebunden werden.

Zunächst informell fortentwickelt von einer Gruppe von Anwendern, wurde OWL im Jahr 2012 schließlich als OWL2 als offizielle Empfehlung des World Wide Web Consortiums (W3C) herausgegeben. Eine Vielzahl von Features wurden verbessert und hinzugefügt.

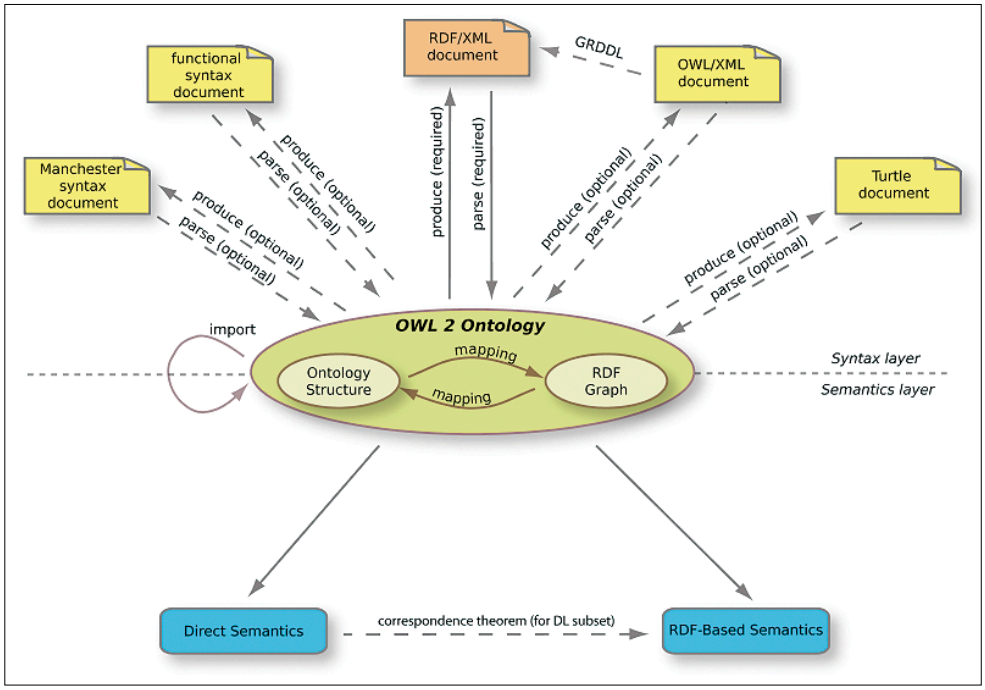
So wurde OWL2 nicht nur mit neuen Funktionen zur Verwaltung von Fakten und Restriktionen, sogenannter Axiome, ausgerüstet, sondern auch die Mechanismen für logische Schlussfolgerungen in den sogenannten Reasonern wurden erweitert und beschleunigt. Und damit etablierten sich die Ontologien schließlich zusammen mit den ebenfalls aufkommenden Graphdatenbanken zu neuen Technologien für das Wissensmanagement – Tendenz stetig steigend.

Ein wichtiger Aspekt von Wissensdatenbanken ist, dass sie Daten und Strukturen standardisieren und damit die Interoperabilität zwischen Systemen unterstützen. Durch ihre Semantik können Ontologien von verschiedenen Systemen gelesen, verstanden und somit in heterogenen Umgebungen als zentrale Ressource gemeinsam genutzt werden.

Was aber die semantischen Web-Ontologien erst zur Grundlage smarter Applikationen macht, ist insbesondere ihre Fähigkeit zu logischen Schlussfolgerungen, basierend auf bereits bestehendem Wissen, die sogenannte Inferenz. Verantwortlich für die Inferenzprozesse sind die Reasoner. Ihre wesentlichen Aufgaben sind es, Objekte – in Ontologien Individuals genannt – zu klassifizieren, automatisiert neues Wissen zu generieren sowie Inkonsistenzen innerhalb von Wissensdatenbanken aufzudecken.

All dies geschieht im sogenannten Klassifizierungsprozess des Reasoners. Dieser umfasst die Erzeugung und die Aktualisierung des Graphen, der im Wesentlichen aus den zwei Bereichen A-Box und T-Box besteht. Während die T-Box alle Konzepte und Klassen, die Terminology Axioms, umfasst, sind in der A-Box alle Assertion Axioms, die eigentlichen Instanzen und Objekte, hinterlegt.

Bezüglich aller Entitäten in einer Ontologie gilt es zu beachten, dass die aus der objektorientierten Programmierung (OOP) bekannte Unique Name Assumption hier nicht gilt.



OWL2: Syntax und Semantik-Schicht (Bild 1)

Diese besagt, dass die Namen von Klassen innerhalb eines Geltungsbereichs immer eindeutig sein müssen. In Ontologien hingegen können aber durchaus mehrere Klassen, Objekte oder Properties mit demselben Namen existieren. Um diese dennoch unterscheiden zu können, wurde für Ontologien der sogenannte Internationalized Resource Identifier (IRI) eingeführt – ein Namespace, der als Prefix, durch ein Hash # getrennt, dem eigentlichen Bezeichner direkt oder als Alias vorangestellt wird:

Anonyme Klassen und Klassenhierarchie (Bild 2)

```
<!-- IRI: http://www.enapso.com/root#Customer -->
<owl:Class rdf:about=
"http://www.enapso.com/root#CustomerCustomer"/>
xmlns:enapso="http://www.enapso.com/root#"
:
<!-- IRI: http://www.enapso.com/root#Customer -->
<owl:Class rdf:about= "&enapso;Customer"/>
```

Mit OWL2-Ontologien können wir großartige Wissensdatenbanken aufbauen – mit Definitionen und Daten, mit Restriktionen und Regeln. Den wesentlichen Teil der Arbeit mit den gespeicherten Informationen leisten dabei die Reasoner. Sie sind es, die letztlich den statischen Konstrukten und Daten Leben einhauchen und Ontologien so zu smarten Datenbanken machen.

Je nach Umfang und Komplexität der in einer Wissensdatenbank umfänglich vernetzten Informationen benötigen die Inferenzprozesse schon mal eine gehörige Portion Rechenleistung und Speicherressourcen. Hier kommt ein Begriff ins Spiel, der Ihnen in der Welt des semantischen Webs häufig begegnen wird: die sogenannte Tractability – die Lösbarkeit eines Problems in einer vertretbaren Zeit.

Im schlimmsten Fall kann ein Problem sogar intractable, also nicht innerhalb einer bestimmten Zeit oder sogar gar nicht lösbar sein – denken Sie beispielsweise an eine Ontologie mit der Komplexität einer natürlichen Sprache. Allerdings können verschiedene Metriken wie zum Beispiel die Anzahl bestimmter Axiome die Überwachungsaufgaben erheblich vereinfachen.

Im Lauf dieses Artikels werden wir einige Best Practices vorstellen. Zum einen, um die Lösbarkeit gerade auch während der Entwicklungs- und Testphasen sicherzustellen. Zum anderen aber auch, um dauerhaft eine angemessene Performance der Reasoner während ihrer Klassifizierungs- und Inferenzprozesse sowie der Abfragen zu gewährleisten.

Ontologien und Reasoner kooperieren eng, arbeiten aber grundsätzlich getrennt voneinander. Während die Ontologien die statischen Entitäten wie Klassen, Objekte und Properties enthalten, übernehmen die Reasoner die funktionalen Aufgaben und sorgen damit für die Dynamik einer Ontologie. Bekannte Reasoner sind HermiT, Pellet oder TrOWL. Sie unterscheiden sich in Performance, Ressourcennutzung, den verwendeten logischen Algorithmen sowie im Funktionsumfang, beispielsweise dadurch, welche Axiome verarbeitet werden können.

OWL2-Ontologien modellieren

OWL2 in Verbindung mit einem Reasoner ist eine leistungsfähige Plattform, um nahezu alle Dinge der realen Welt sowie deren Beziehungen und Abhängigkeiten abzubilden. Ontologien werden prädestiniert dort eingesetzt, wo Informationen in vernetzten und hierarchischen Baumstrukturen organisiert werden und damit zu Klassenmodellen (Taxonomien) einer bestimmten Domäne werden. Eine klassische Taxonomie ist eine Personalhierarchie – beispielsweise die eines Unternehmens.

Ontologien sind jedoch weit mehr als reine Taxonomien. Sie sind Graphen, in denen die einzelnen Knoten untereinander vernetzt sind. Veranschaulichen wir dies am Beispiel einer Universität mit mehreren Fakultäten – und natürlich auch mit einer Personalhierarchie. Deren Struktur, also die eigentlichen Job-Positionen, ist abgelegt in der T-Box, dem Klassendiagramm, die Personen dahinter als Individuen innerhalb der A-Box. Nun stellen Sie sich die Menge an Kursen verteilt über die Fakultäten vor, die benötigten Ressourcen für einen Kurs und die Teilnehmer der verschiedenen Kurse: Eine Vielzahl an Verlinkungen, die in klassischen SQL-Datenbanken mit einer ebenso großen Vielzahl von n:m-Relationen abgebildet werden müssen – ganz zu schweigen davon, diese auch unabhängig voneinander zu warten.

In einem Graphen dagegen werden lediglich atomare Informationen abgelegt und deren Verlinkungen werden nur einmal pro Entität persistiert. Damit reduziert sich der Wartungsaufwand, die Übersicht verbessert sich und strukturbedingte Redundanzen werden vermieden.

Mit dem traditionellen Modell der objektorientierten Programmierung werden Klassenmodelle seit vielen Jahren mit Baumstrukturen assoziiert und auch so abgebildet. Man mag daher versucht sein, auch bei dem Begriff Ontologie zunächst nur an Klassenmodelle mit ihren bekannten Generalisierungs- und Spezialisierungsbeziehungen zu denken.

Ontologien bieten allerdings wesentlich mehr, insbesondere in Bezug auf Standardisierungen und Ausdrucksstärke. Zwar können in Ontologien ebenso wie in Taxonomien auch Klassenbäume definiert werden, darüber hinaus aber auch Beziehungen zwischen Klassen untereinander sowie die Zuordnung von Individuals zu einer oder mehreren Klassen. Ontologien unterstützen grundsätzlich Mehrfachvererbung.

Weiterhin können Sie mit Restriktionen die Inhalte Ihrer Wissensdatenbanken kontrollieren. Beispielsweise in Form von Aussagen, die wahr sein müssen, um weitere Aussagen zuzulassen, oder auch, um Werte für Eigenschaften auf einen bestimmten Datentyp oder bestimmte Wertebereiche zu beschränken.

Semantische Regeln (SWRL)

Ein weiteres leistungsfähiges Feature zur Modellierung von OWL2-Ontologien sind Regeln, formuliert in der Semantic ▶

Web Rule Language (SWRL). Sie ermöglichen in der Form *Bedingung => Konsequenz*, in anderen Worten auch *if (condition) then { consequence }*, während des Inferenzprozesses automatisch neue Fakten – und damit neues Wissen – zu generieren.

Stellen Sie sich vor, in der Domäne Ihrer Ontologie wäre jede Person älter als 18 Jahre berechtigt, ein Auto zu fahren, und die Klasse *LicensedDriver* würde hierzu alle Personen über 18 Jahre beschreiben. Erinnern Sie sich, dass jedes Individual – hier also jede Person – mehreren Klassen zugeordnet sein kann. Eine Abfrage aller Individuals der Klasse *LicensedDriver* würde also alle Personen ausweisen, die ein Auto fahren dürfen. Das Problem an dieser Stelle ist, dass Personen nicht per se bei Erreichen des 18-ten Lebensjahres der Klasse *LicensedDriver* zugeordnet werden. In der klassischen Datenbank-Technologie müsste man sich hier aufwendiger Trigger und Stored Procedures oder gar applikationsseitiger Background-Threads bedienen, um die entsprechenden Flags korrekt zu persistieren, um diese damit letztlich wieder abfragbar zu machen. In Ontologien geht dies wesentlich einfacher mit einer Regel – und zwar ohne jeglichen Zwang, die Daten selbst persistieren zu müssen. Hierzu dient die folgende SWRL-Direktive:

```
Person(?x)
  ^ hasAge(?x, ?age
  ^ swrlb:greaterThanOrEqual(?age, 18)
  -> LicensedDriver(?x)
```

Der vordere Abschnitt dieser Regel bis zum Pfeil ist der Bedingungsteil. Der Reasoner sucht hier nach allen Personen *?x*, die in der Property *hasAge* einen Wert *?age* haben und dieser größer oder gleich *18* ist. Falls diese Bedingung für das Individual *?x* zutrifft, wird dieses als *LicensedDriver* klassifiziert und damit auch in allen Queries als solches zurückgeliefert – wie schon gesagt: Ohne dass dies explizit so gespeichert ist. Beachten Sie, dass im Bereich der Reasoner mit vielen Investitionen regelmäßig neue technologische Verbesserungen eingebracht werden. Noch nicht alle Reasoner unterstützen bereits SWRL oder deren vollen Sprachumfang. Hier ist aber sicher schon bald mit umfänglichen Innovationen zu rechnen.

Tabelle 1: Grundlegende Konzepte und Begriffe

Klassen	Klassen sind Mengen (Sets oder Collections) von Objekten, im Rahmen von Ontologien also Mengen von Individuals. Beispielsweise die Klasse *Vehicle* als Menge aller Fahrzeuge. Klassen beschreiben den Typ von Objekten, eine Art abstraktes Objekt, dessen Kriterien und Restriktionen für alle Objekte des gleichen Typs gelten.
Individuals	Individuals sind konkrete Instanzen einer Klasse in der Wissensdatenbank. Beispielsweise *Osvaldo* und *Alex* sowie *Osvaldo's Ferrari* und *Alex's Lamborghini* sind Individuals, hier also Instanzen der Klassen *Person* und *Vehicle* – wenn auch in Bezug auf die Fahrzeuge leider nur imaginäre. Achtgeben sollte man auf deren Namen und damit auf die Lesbarkeit der Ontologie. Bei dem Bezeichner *Maserati könnte* es sich zwar um eine konkrete Instanz handeln. Allerdings wäre diese von anderen Instanzen des gleichen Typs dann nur durch ihren IRI zu unterscheiden und könnte eher für eine Klasse gehalten werden. Bedenken Sie in diesem Zusammenhang, dass Ontologien spezialisiert sind auf semantische Anforderungen. Zur Verwaltung umfangreicher Massendaten empfehlen sich möglicherweise hybride Konzepte, auf die wir in einem zukünftigen Artikel in diesem Magazin noch näher eingehen werden.

Properties	Properties repräsentieren Beziehungen, die wichtigsten in OWL sind *Object Properties* und *Data Properties*. *Object Properties* werden verwendet, um Verbindungen zwischen Individuals zu beschreiben. Beispiel:
	
	`ObjectPropertyAssertion(hasCar "Osvaldo's Ferrari" Osvaldo).`
	
	Data Properties hingegen beschreiben die Zuordnung von Literalen zu Individuals. Beispiel:
	
	`DataPropertyAssertion(hasAge Osvaldo 29)`

Annotations	Annotations dienen dazu, zu jeder Entität einer Ontologie Anmerkungen hinzuzufügen. Jede Annotation besteht aus einer Annotation-Property und einem Annotation-Wert. OWL2 kommt bereits mit einigen eingebauten Annotations, beispielsweise *label, comment* oder *versionInfo*, aber Sie können beliebige eigene Anmerkungen hinzufügen, um Ihre Ontologien mit zusätzlichen Informationen zu versehen und sie damit leichter wartbar zu machen.

OWL2-Ontologie-Konzepte

Die wichtigsten Konzepte zum Design von Ontologien sind Klassen (Classes), Eigenschaften (Properties), Objekte (Individuals) und Datentypen (Data Types). All diese werden auch als Entitäten der Ontologie bezeichnet. Der Begriff der Entität mag für uns Entwickler zu Beginn verwirrend klingen, da wir diese aus der Welt des Object-Relational Mapping (ORM) und der Entity-Relationship-Modelle (ER) eher mit Klassen assoziieren, die reale Dinge in unseren Applikationen beschreiben. Aber in Ontologien wird alles, was über eine ID – den Internationalized Resource Identifier (IRI) – verfügt, als Entität bezeichnet.

Bevor wir nun eine konkrete Ontologie erstellen und uns mit deren Potenzialen beschäftigen, lassen Sie uns zuvor noch einige grundlegenden Konzepte und Begriffe erläutern. Tabelle 1 gibt einen Überblick dazu.

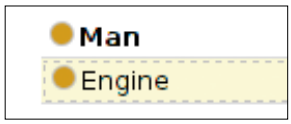
OWL2-Axiome

Die zentrale Komponente, um Wissen in OWL2 zu repräsentieren, sind Axiome. Sie beschreiben die Fakten über *Classes, Individuals* und *Properties* einer Domäne. Neben peripheren Informationen wie beispielsweise Headern, Import-Direktiven oder Prefixes (Aliases für IRIs) bestehen Ontologien im Wesentlichen aus einem Satz von Axiomen und Theoremen.

Während Axiome als allgemein akzeptierte und wahre Aussagen definiert sind, sind Theoreme Aussagen, die auf Basis von Axiomen oder anderen Theoremen bewiesen worden sind. Beide zusammengenommen geben den Objekten in einer Ontologie ihre semantische Bedeutung. Allerdings

werden sie dort lediglich unter dem Sammelbegriff Axiome geführt.

Es sind genau diese Axiome, die es uns Menschen ebenso wie den Reasonern und damit unseren Applikationen erlauben, das Wissen in einer Ontologie zu verstehen. Axiome machen Aussagen darüber, was richtig oder falsch ist. Sie beschreiben Daten und Beziehungen zwischen Entitäten ebenso wie Restriktionen und Regeln. Sie sind die Grundlage für die Reasoner, um neues Wissen logisch zu erschließen, welches der Ontologie nie explizit vermittelt wurde. Dieser Vorgang des Schlussfolgerns, der Inferenzprozess, bei dem neues Wissen auf Basis von bestehendem Wissen maschinell generiert wird, ist einer der Eckpfeiler zur Entwicklung smarter Applikationen.



Falsche Schlussfolgerung, dass Mike auch ein Motor ist (Bild 3)

Erzeugen der Wissensbasis

Nachdem wir nun die Hauptkomponenten, die Entitäten von OWL2-Ontologien sowie deren Axiome, kennengelernt haben, lassen Sie uns nun einige Beispiele betrachten, um eine bessere Vorstellung davon zu bekommen, wie wir konkret vom Wissen in unseren Ontologien profitieren können.

Ontologien sind nicht nur geeignet, von Maschinen verstanden zu werden, sondern sie wurden genau dazu geschaffen. Im Umkehrschluss bedeutetet das allerdings, dass sie nun von uns Menschen, beispielsweise in Form von XML-Dateien, nicht mehr so leicht lesbar sind. Glücklicherweise bestehen Ontologien aber aus zwei Schichten, einer syntaktischen und einer semantischen. Die syntaktische ist unter anderem durch die Manchester-Syntax oder auch die Functional-Style-Syntax realisiert, welche beide untereinander austauschbar und leicht verständlich sind. Bild 1 bietet Ihnen einen besseren Überblick über diese Strukturen in OWL2.

In den Beispielen in diesem Artikel verwenden wir konsequent die Functional-Style-Syntax, da wir diese für einfacher nachvollziehbar halten. Lassen Sie uns also einen Blick auf ein erstes Klassenmodell werfen, eine simple Taxonomie in einer Ontologie über Fahrzeuge:

```
Zeile 1: SubClassOf( Car Vehicle )
Zeile 2: SubClassOf( Ferrari Car )
```

Diese zwei einfachen Axiome beschreiben eine hierarchische Struktur, in der *Car* eine Unterklasse von *Vehicle* und *Ferrari* eine Unterklasse von *Car* ist. Hier sehen Sie bereits, wie einfach es ist, ein Klassifizierungsschema in einer Ontologie aufzubauen. Wichtiger Unterschied zu einer Datenbank ist, dass die Aussage *Ferrari ist eine Unterklasse von Vehicle* nicht explizit gespeichert sein muss, um als solche abgefragt werden zu können.

Natürlich könnten wir diese Informationen auch in einer Datenbank speichern und die Vererbung emulieren. Aber Datenbanken sind spezialisiert auf Daten und performanceoptimiert in Bezug auf Abfragen und Aktualisierungen – nicht jedoch auf Ausdrucksstärke oder gar Inferenzen innerhalb eines Schemas. In anderen Worten: Daten, die nicht explizit in einer Datenbank hinterlegt worden sind, können aus dieser auch nicht abgefragt werden – ganz im Gegensatz zu Ontologien.

Assertions

Nachdem wir nun ein erstes Klassenmodell angelegt haben, lassen Sie uns als Nächstes sehen, wie wir Individuals und Properties zu unserer Ontologie hinzufügen:

```
Zeile 3: ClassAssertion( Mike Person )
Zeile 4: DataPropertyAssertion( hasAge Mike 40 )
Zeile 5: ClassAssertion( "Mike's Maserati" Maserati )
Zeile 6: ObjectPropertyAssertion( hasCar Mike "Mike's
Maserati" )
```

Die obigen Axiome zeigen, wie eine Wissensdatenbank gefüllt wird mit Aussagen über Individuals, den sogenannten Assertions. Im Gegensatz zu klassischen Datenbanken definiert ein Ontologie-Schema keine Tabellen mit Zeilen und Spalten oder Kollektionen mit Dokumenten. Vielmehr erhält jede Instanz Daten ähnlich dem Modell der objektorientierten Programmierung.

Die sogenannten ClassAssertions beschreiben die Zuordnung von Individuals zu Klassen. Die obigen Zeilen 3 und 5 definieren beispielsweise, dass *Mike* zur Klasse *Person* gehört sowie dass die konkrete Instanz *Mike's Maserati* zur Klasse *Maserati* gehört.

Weitere Assertions sind *DataPropertyAssertions* sowie *ObjectPropertyAssertions*. In der Zeile 4 im obigen Listing sagt die Data Property Assertion *hasAge*, dass Mike ein Alter von *40* hat – wobei wir die Einheit *Jahre* für das Literal 40 bisher nur vermuten, aber noch nicht wissen. Die Object Property Assertion *hasCar* in der Zeile 6 hingegen sagt aus, dass Mike das Auto *Mike's Maserati* besitzt – also die Beschreibung einer Relation zwischen zwei Individuals.

Beachten Sie, dass wir bei den Property Assertions zwar Individuals mit echten Daten erzeugen, aber keinerlei strukturelle Änderungen am Klassenmodell vornehmen. Im Gegensatz zu Tabellen-Strukturen kann jedes Individual in einer Ontologie eigene Properties, Daten und Relationen enthalten – insbesondere völlig unabhängig von seinen Klassenzuordnungen.

Property Restrictions

Auf den ersten Blick mag es erscheinen, als handele es sich bei den Ontologien um nicht mehr als eine andere, vielleicht sogar komplexere Art der Datenverwaltung. Aber die Property Assertions leisten wesentlich mehr, als nur Werte und Verknüpfungen zu verwalten. Als Entitäten innerhalb der Wissensdatenbank repräsentieren diese die eigentlichen semantischen Bedeutungen, wie die folgenden Axiome zeigen:

```
Zeile 7: ObjectPropertyDomain( hasEngine Vehicle )
Zeile 8: ObjectPropertyRange( hasEngine Engine )
Zeile 9: FunctionalDataProperty( hasAge )
```

▶

Tabelle 2: Wichtige Aspekte

1	**Vorsicht bei Domains und Ranges für Properties** Unbedacht definierte Domains und Ranges für Properties können insbesondere bei größeren Ontologien leicht zu Inkonsistenzen führen. Weiter unten in diesem Artikel finden Sie konkrete Beispiele hierzu. Am besten dokumentieren Sie alle Restriktionen sorgfältig mit Hilfe von Annotations, um unerwünschte Inferenz-Seiteneffekte später leicht korrigieren zu können.
2	**Besser some- als only-Restrictions verwenden** In einigen Fällen ist es besser, die existential (some) als die universal (only) Restriction zu verwenden, und zwar aus demselben Grund wie bei den Domain- und Range-Restrictions. only macht die Ontologie sehr restriktiv. Stellen Sie sich die only-Restriktion *hasWife only Women* vor. Im Fall einer gleichgeschlechtlichen Ehe würde dann aus *Peter hasWife Paul* automatisch schlusszufolgern sein, dass Paul eine Frau sein muss. In anderen Worten: Seien Sie sich beim Einsatz zu starker Restriktionen wirklich sicher. Was in kleinen Ontologien noch kein allzu großes Problem darstellt, kann zur Notwendigkeit weitreicher Änderungen führen.
3	**Vermeiden Sie vorgefertigte Design Patterns** Zwar können vorgefertigte Ontology Design Patterns (ODPs) zur schnellen Implementierung einer Lösung für bestimmte Probleme beitragen. Aber: ODPs sind häufig optimiert in Bezug auf Ausdrucksstärke, die Expressivity, was mit einer Vielzahl von Axiomen einhergeht, die Sie in für Ihren konkreten Einsatz möglicherweise gar nicht benötigen. ODPs eignen sich sehr gut zu Trainingszwecken und als Beispiele, sollten jedoch nicht einfach blind übernommen werden. Im schlimmsten Fall können sie unerwünschte und später nur schwer nachvollziehbare Effekte für Ihre Ontologie bedeuten. Den Nachteil spüren Sie deutlich, sobald die Ontologie wächst und die Reasoner dann durch die Menge an Axiomen an Performanz verlieren.
4	**Vermeiden Sie Performance-Fresser** Vermeiden Sie, wenn nicht zwingend erforderlich, Features wie disjunctions, transitive Object Properties, disjoints oder inverse Objekt-Beziehungen, um die Reasoner- und Query-Performance Ihrer Ontologie zu verbessern. Inverse Properties wie beispielsweise hasCar zusammen mit *isCarOf* kosten Performance, weil die umgekehrte Property immer neu generiert werden muss – und das bedeutet einen hohen Aufwand für den Reasoner bei großen Ontologien. Ähnliches gilt für transitive Properties: Auch hier müssen gegebenenfalls viele neuen Verlinkungen geschaffen werden. Die Regel hier ist: Höhere Ausdrucksstärke der Ontologie geht auf Kosten der Performance. Die Kunst ist die perfekte Balance.
5	**Auswahl des optimalen OWL2 Profils** OWL2 offeriert einige sogenannte Profile, syntaktische Untermengen von OWL2, die zwar funktionale Einschränkungen, dafür aber verschiedene Vorteile abhängig vom jeweiligen Einsatzgebiet Ihrer Ontologie bedeuten. Die verfügbaren Profile sind OWL2 EL, OWL2 QL und OWL2 RL. Jedes Profil ist eine Untermenge des OWL2-Sprachumfangs und restriktiver als OWL2. Alle bieten zwar geringere Ausdrucksstärke, dafür aber höhere Performance. OWL2 EL ist optimiert für Ontologien mit einer hohen Anzahl an Klassen beziehungsweise Properties. Der hauptsächliche Einsatz ist, umfangreiche Domänen-Modelle zu beschreiben, und zwar mit einer eher geringen Anzahl Instanzen. OWL2 QL hingegen zielt eher auf Ontologien ab, bei denen es auf die schnelle Beantwortung von Queries bei einer eher großen Menge von Instanzen ankommt. OWL2 QL wurde insbesondere für die einfachere Wartung und Abfrage von Daten in Wissensdatenbanken designt. OWL2 RL schließlich wurde spezifiziert für Anwendungen bei denen es auf die skalierbare Ausführung der Inferenzprozesse, also der Reasoner, ankommt, ohne dabei allzu viel Ausdrucksstärke zu opfern.

Hier werden die einfachen Konzepte Domain und Range auf Properties appliziert, um Restriktionen zu definieren, aber auch, um damit weiteres Wissen erzeugen zu können.

Diese beiden mathematischen Konzepte stammen aus der Set-Theorie, die wir hier nur grob anreißen können. Vereinfacht gesagt definiert die Domain einer Property die Elemente der linken Seiten einer Relation und die Range die Menge der Werte, die auf der rechten Seite einer Relation auftreten können.

Beispiel: Durch die Deklaration der Axiome 7 und 8 im oberen Listing können wir schlussfolgern, dass innerhalb der Domäne unserer Ontologien alle Individuen, die einen Motor haben, Instanzen der Klasse *Vehicle* sein müssen, und dass alle durch *hasEngine* referenzierten Instanzen ein Motor sein müssen.

Properties können außerdem *functional* sein, wie im obigen Listing in der Zeile 9. Das bedeutet, dass ein Individual nur eine einzige dieser Eigenschaft zugewiesen werden kann. Während verständlicherweise eine Person nur ein Alter haben kann, könnte diese aber dennoch mehrere Autos besitzen. Weitere Charakteristiken von Properties sind: *transitive, symmetric, reflexive* und noch einige weitere, die die semantische Bedeutung von Properties noch erweitern.

Transitive Properties unterstützen zum Beispiel Ketten von Beziehungen. Nehmen Sie exemplarisch die transitive Property *subRegionOf*: Ist *B* eine *subRegionOf A* und ist *C* eine *subRegionOf B*, dann gilt, dass auch *C* eine *subRegionOf A* ist. Ein klassisches Beispiel für eine symmetrische Property ist hingegen *friendOf*: Ist *A friendOf B*, dann gilt automatisch auch *B friendOf A* – ohne dass diese Aussage explizit hinterlegt wurde.

Auf Ontologien basierende Wissensdatenbanken sind derart ausdrucksstark, dass einige Ontologien sogar ausschließlich aus Taxonomien (Klassenmodellen), Properties und Restriktionen bestehen und so komplexe Domänen ohne ein einziges Individual beschreiben. Sehr effektiv können dann hybride Ansätze, beispielsweise in Kombination mit NoSQL-Datenbanken, die semantischen Vorteile der Ontologien mit der Performance von Big-Data-Ansätzen verbinden.

Class Restrictions

Weitere häufig in Ontologien verwendete Restriktionen sind *universal (only)* und *existential (some)*. Diese werden eingesetzt, um Modelle genauer zu beschreiben. Lassen Sie uns anhand eines Beispiels zeigen, wie diese Restriktionen wirken und wie wir sie nutzen können:

```
Zeile 10: SubClassOf( Vehicle ( hasEngine only Engine )
)
Zeile 11: SubClassOf( Person ( hasCar some Car ) )
```

Das Axiom in der Zeile 10 bedeutet, dass die Klasse *Vehicle* nun eine Unterklasse einer anonymen und namenlosen Klasse mit der Restriktion *(hasEngine only Engine)* ist. Dies bewirkt, dass für Instanzen der Klasse *Vehicle* und die Property *hasEngine* ausschließlich Individuals der Klasse *Engine* als konsistent akzeptiert werden.

Das Axiom in der Zeile 11 hingegen sagt aus, dass Individuals der Klasse *Person* über die Property *hasCar* mit *einigen* Individuals der Klasse *Car* vernetzt sein kann. Eine Person kann daher kein, ein einzelnes oder mehrere Autos besitzen.

Der generelle Aspekt hier ist, wie das bestehende Wissen über unsere Domäne auf der Klassenebene repräsentiert wird. Bild 2 zeigt, was wir konkret mit den beiden Axiomen aus den Zeilen 10. und 11. in der Ontologie erzeugt haben.

Die für die Restriktion erzeugte anonyme Klasse, in Bild 2 in Grün dargestellt, sorgt dafür, dass *Vehicle* als ihre Unterklasse ebenso wie *Car* und *Maserati* als weitere Unterklassen von *Vehicle* diese Restriktionen erben. Auf diese Art und Weise werden Restriktionen durch den Klassenbaum propagiert, was Ontologien mit einem leistungsfähigen Vererbungs- und Constraint-Feature ausstattet, das man bei herkömmlichen Datenbanken vergeblich sucht.

Der Sprachumfang von OWL2 enthält eine Vielzahl weiterer Axiome und Konzepte, die sicher den Umfang dieses Artikels sprengen würden. Wenn Sie bereits planen, Ontologien in Ihren Applikationen einzusetzen, sei Ihnen zum einen die W3C-Referenz-Dokumentation empfohlen. Zum anderen können Sie aber auch schon praktische Erfahrungen sammeln mit Protégé, dem derzeit wohl am weitesten verbreiteten freien Ontologie-Management-Tool.

Mit Hilfe einer Vielzahl von Plug-ins können Sie dort auch verschiedene Reasoner integrieren, DL- oder SPARQL-Queries ausführen oder regelbasierte Technologien wie SWRL anwenden. In den nächsten Ausgaben der **web & mobile developer** werden wir weiter ausführlich über diese Themen berichten.

Best Practices

Als Entwickler mögen Sie zum Einstieg nach Best Practices und einigen Design-Patterns für Ontologien fragen. Grundsätzlich gibt es zunächst einmal zwei wesentliche Aspekte, die man zum Design performanter Ontologien von Beginn an im Auge haben sollte: die Lösbarkeit (Tractability), und die Fähigkeit des Modells, konsistent zu bleiben, also in sich schlüssig und widerspruchsfrei zu bleiben, insbesondere dann, wenn die Ontologie wächst.

OWL2 stellt eine Vielzahl von Möglichkeiten bereit, um nahezu alle Dinge der realen Welt abzubilden. Der beste Weg, ein Model vom individuellen Wissen ihrer konkreten Domäne zu erstellen, hängt jedoch von verschiedenen Faktoren ab. Natürlich ist die erwartete Größe der Ontologie ein solcher, aber insbesondere die perfekte Balance zu finden zwischen hoher Performance oder hoher Ausdrucksstärke auf der einen Seite sowie der Menge an explizitem Wissen oder intensiver Inferenzaufgaben für den Reasoner auf der anderen Seite, stellt zuweilen eine Herausforderung dar. In Tabelle 2 wollen wir Ihnen daher einige wichtige Aspekte rund um diese Entscheidungen mit auf den Weg geben.

Reasoner

Wie wir bis hierher schon erfahren haben, spielen die Reasoner eine zentrale Rolle in Wissensdatenbanken. Nicht nur, weil sie während des Inferenzprozesses für die Generierung neuen Wissens basierend auf Axiomen und Regeln verantwortlich sind. Auch während des Entwicklungsprozesses für eine Ontologie dienen sie als wertvolles Werkzeug, um ihre Konsistenz und Erfüllbarkeit zu überprüfen. Sie erklären in nachvollziehbarer Weise, welche Axiome für automatisches generiertes Wissen, also Schlussfolgerungen, herangezogen wurden, und ebenso, auf Basis welcher Widersprüche eine Ontologie inkonsistent wurde.

```
1) Mike Type Man
2) Maserati hasEngine Mike
3) hasEngine Range Engine
4) Engine DisjointWith Man
```

Ausweis von Inkonsistenzen in einer Ontologie (Bild 4)

In Bild 3 sehen Sie, wie aufgrund der Open World Assumption (OWA) und des Axioms aus Zeile 12 durch den Reasoner das Individual *Mike* als Mann und fälschlicherweise ebenso als Motor klassifiziert wird:

Bezüglich der Erfüllbarkeit einer Ontologie stellen Sie sich die Klasse *Airplane* als Nachfahre von *Vehicle* vor und verhindern Sie mittels einer *DisjointClasses* Direktive, dass ein Individual gleichzeitig ein Auto und ein Flugzeug sein kann. Dies wird sicher eine Weile gutgehen, bis zu dem Tag, an dem die Autos nicht nur selbst fahren, sondern vielleicht mit Flügeln auch die dritte Dimension erobern. Ein Individual der Klasse *FlyingCar* wäre dann unerfüllbar.

Open World Assumption

Lassen Sie uns eine spezielle Charakteristik von Ontologien im Allgemeinen und der Reasoner im Speziellen an einem Beispiel verdeutlichen. Hierzu fügen wir das folgende Axiom zur unserer Ontologie hinzu und starten den Reasoner:

```
Zeile 12: ObjectPropertyAssertion(
  hasEngine "Mike's Maserati" Mike
)
```

Das Ergebnis dieses Axioms ist erst einmal genau so, wie die Aussage es beschreibt, nämlich dass das Individual *Mike's Maserati* das Individual *Mike* als *Engine* hat. Was sich hier zunächst als simple Fehleingabe darstellt, kann in einer Ontologie aber leicht zu weitreichenden Konsequenzen führen. Denn sobald Sie den Reasoner starten, schlussfolgert dieser, dass Mike ein Mann, aber ebenfalls ein Motor ist. Wenn Sie sich nun wundern warum, schauen Sie auf das Axiom in Zeile 8. Dort ist für die Property *hasEngine* eine Range spezifiziert, die besagt, dass alle von ihr referenzierten Individuals vom Typ *Engine* sind.

An dieser Stelle sei nochmals deutlich darauf hingewiesen, dass es sich in Ontologien um Aussagen und nicht um harte Datenbank-Constraints handelt. Ontologien lassen die Aufnahme von Aussagen erst einmal zu, und die Reasoner übernehmen anschließend die logischen Inferenz-Aufgaben.

Verantwortlich dafür ist das OWL2-Paradigma, dass Ontologien und Reasoner auf Basis der Open World Assump- ▶

Links zum Thema

- Enapso Smart Solution Projekt
  https://enapso.org
  https://www.innotrade.com
- Description logic (Wikipedia)
  https://en.wikipedia.org/wiki/Description_logic
- Resource Description Framework (RDF)
  www.w3.org/RDF
- Internationalized Resource Identifiers (IRIs)
  https://www.ietf.org/rfc/rfc3987.txt
- Objektrelationale Abbildung (Wikipedia)
  https://de.wikipedia.org/wiki/Objektrelationale_Abbildung
- Entity-Relationship-Modell (Wikipedia)
  https://de.wikipedia.org/wiki/Entity-Relationship-Modell
- OWL 2 Web Ontology Language – Manchester Syntax
  www.w3.org/TR/owl2-manchester-syntax
- OWL 2 Web Ontology Language – Structural Specification and Functional-Style-Syntax
  www.w3.org/TR/owl2-syntax
- OWL 2 Web Ontology Language
  www.w3.org/TR/owl2-overview
- protégé – Stanford University
  http://protege.stanford.edu
- SPARQL Query Language for RDF
  www.w3.org/TR/rdf-sparql-query
- Ontology Design Patterns
  http://ontologydesignpatterns.org/wiki/Main_Page

tion (OWA) arbeiten. Diese besagt vereinfacht ausgedrückt, dass eine fehlende Aussage nicht automatisch bedeutet, dass sie falsch ist, im Umkehrschluss also auch, dass eine Aussage durchaus wahr sein kann, auch wenn sie nicht ausdrücklich als solche angegeben wurde. Nur der Vollständigkeit halber: Die Closed World Assumption (CWA) besagt, dass alles, was nicht ausdrücklich als wahr bekannt ist, falsch sein muss – auch wenn dies in Ontologien nicht zur Anwendung kommt.

Aber zurück zu unserem Beispiel: Da wir definieren, dass die Range der Property *hasEngine* immer eine *Engine* ist, klassifiziert der Reasoner bei der Analyse des Axioms in Zeile 12 Mike als Motor.

Und genau dieser Effekt möglicherweise unzutreffender Inferenzen ist der Grund, warum Sie Domains und Ranges für Properties mit Bedacht einsetzen sollten, insbesondere dann, wenn Sie den Aufbau umfangreicherer Ontologien planen.

Inkonsistenzen in Ontologien

Ontologien werden dann inkonsistent, wenn der Reasoner aufgrund von Widersprüchen zwischen verschiedenen Axiomen keine neuen Fakten mehr für eine Ontologie, also kein neues Wissen mehr generieren kann, was für eine Ontologie ein Ausschlusskriterium darstellt.

Beispiel: Belassen Sie die bisherigen Axiome in der Ontologie, ergänzen Sie das nachfolgende Axiom und lassen Sie den Reasoner erneut laufen:

```
DisjointClasses( Man Engine )
```

Der Reasoner wird nun ausweisen, dass die Ontologie inkonsistent ist (Bild 4).

Der Reasoner stellt eine Erklärung für die Inkonsistenz bereit. Hier sagt er: »Sie sagen, dass Mike ein Mann ist, dass Mike der Motor eines Maseratis sei, dass aber *hasEngine* nur Motoren beschreiben kann und dass Motoren und Männer nicht das Gleiche sind« – das ist eindeutig eine logische Inkonsistenz.

Fazit

Das Potenzial von OWL2-Ontologien ist einfach riesig: Sie vereinen Ausdrucksstärke für eine semantische Wissensrepräsentation mit einem standardisierten und maschinenlesbaren Format und sind darüber hinaus bereits in einer Vielzahl von wirtschaftlichen und sozialen Anwendungsfeldern etabliert.

Die Reasoner ebenso wie die Abfrage- und Regelsprachen machen OWL2 nicht nur zur zentralen Technologie für das semantische Web, sondern ebenso zur Grundlage von Anwendungs- und sektorübergreifenden Wissensdatenbanken.

Für uns als Software-Entwickler sind Ontologien die Basis für neue Lösungen rund um Smart Data. Sie verbessern die Qualität unserer Daten durch Konsistenz, unterstützen Semantik in deren Modellen und Abfragen, vereinfachen deren Wartung und erlauben die automatische, aber kontrollierte Generierung von neuem Wissen.

Es lohnt sich also, sich mit Ontologien und OWL2 auseinanderzusetzen, neue Applikationen smart zu machen und damit letztlich die User-Experience zu verbessern. In den nächsten Ausgaben der **web & mobile developer** werden wir Sie gerne weiter auf dem Laufenden halten. ■

Osvaldo Aguilar Lauzurique
ist Informatik-Ingenieur mit über zehn Jahren Erfahrung in den Bereichen Webentwicklung, künstliche Intelligenz, Software-Architekturen und Datenbankdesign. Er ist aktiver jWebSocket-Committer.

Alexander Schulze
ist Software-Architekt und IT-Consultant mit über 25 Jahren Erfahrung. Er ist Gründer von jWebSocket, Fachautor und Sprecher auf internationalen Konferenzen.
www.innotrade.com

developer media

Katalog 2016 unter developer-media.de

Katalog herunterladen und Fortbildung auswählen

developer media T Trainings

Praxisorientierte Präsenztrainings

developer media W Webinare

Bequem lernen ohne Reisen

developer media D Derbooks

Themenrelevantes Wissen

developer media A Apps

Lernen von unterwegs

developer media E Events

Austausch mit Experten vor Ort

developer media M Magazine

Monatliche Updates

Ihr Ansprechpartner:
Fernando Schneider – Key Account Manager – developer media
Telefon: +49 (0)89 74117-831 – E-Mail: fernando.schneider@developer-media.de

SICHERE HÄFEN UND GESPEICHERTE VORRATSDATEN

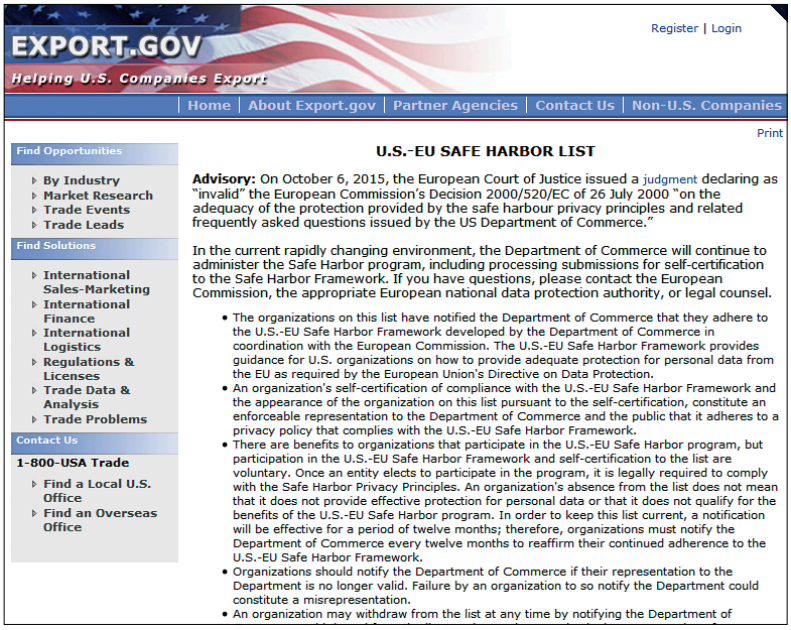
Datenschutz

Diverse Neuerungen in puncto Datenschutz beschäftigen Designer, Programmierer und Betreiber von Websites.

Das IT-Sicherheitsgesetz in Kraft getreten, die Vorratsdatenspeicherung eingeführt, Safe Harbor gekippt – 2015 war in datenschutzrechtlicher Hinsicht ein sehr ereignisreiches und zugleich spannendes Jahr. Und viele Datenschützer sind vermutlich der Meinung, dass der Erlass des IT-Sicherheitsgesetzes ein guter Schritt in die richtige Richtung war, die Vorratsdatenspeicherung jedoch ein Schritt in die Gegenrichtung.

Im gleichen Zeitraum, in denen die deutsche Bundesregierung diese beiden Gesetzespakete auf den Weg gebracht hat, ist es einem Jurastudenten und Datenschutzaktivisten aus Österreich tatsächlich gelungen, sowohl Facebook und die irische Datenschutzaufsichtsbehörde als letztlich auch die gesamte sogenannte Safe-Harbor-Regelung nicht nur zu hinterfragen, sondern diese sogar gänzlich zu Fall zu bringen.

Es gibt also einige wichtige Neuerungen im Datenschutzrecht. Man kann und wird wohl lange über einzelne Details diskutieren – Fakt ist jedoch, dass insbesondere Betreiber, Designer und Programmierer von Internetseiten daran nicht vorbeikommen. Dies betrifft in erster Linie das IT-Sicherheitsgesetz, aber natürlich auch die Safe Harbor-Regelung.



Datenbank mit den bereits für Safe Harbor registrierten Unternehmen (Bild 1)

Sicherer Hafen

Anfang des neuen Jahrtausends, nämlich 2000, hat die Europäische Union mit den USA ein Abkommen zur sicheren Transatlantik-Übertragung persönlicher Daten vereinbart. Die Daten von EU-Bürgern sollten auf dem nordamerikanischen Kontinent einen »sicheren Hafen« finden. Folgerichtig erhielt dieses Abkommen die Bezeichnung Safe Harbor.

Primäres Ziel dieser Regelung war es, bei der Übermittlung von personenbezogenen Daten aus einem EU-Mitgliedsstaat in die Vereinigten Staaten dem hiesigen Datenschutz-Niveau entsprechende Rahmenbedingungen und damit Rechtssicherheit für europäische Unternehmen zu schaffen.

Amerikanische Unternehmen können sich zu diesem Zweck in eine Liste des US-Handelsministeriums eintragen, dies allerdings auf freiwilliger Basis (Bild 1). Dadurch konnte die Einhaltung essenzieller Datenschutzregelungen überprüft und bei etwaigen Verstößen auch sanktioniert werden. Das Safe-Harbor-Abkommen war letztlich als Voraussetzung dafür gedacht, dass Daten mit Personenbezug aus Europa auch ohne ausdrückliche Einwilligung der Betroffenen in die USA transferiert werden konnten; innerhalb Europas war und ist dies hingegen unproblematisch.

Als Alternative zu Safe Harbor können auch die sogenannten EU-Standardvertragsklauseln genutzt werden. Hierbei handelt es sich um exakte vertragliche Vorgaben seitens der EU-Kommission, welche in möglichst unveränderter Form zum Einsatz kommen sollte. International agierende Konzerne können außerdem interne Regelungen (die sogenannten Binding Corporate Rules) formulieren und umsetzen.

Allerdings war die Nutzung des Safe Harbor-Abkommens bis zur Entscheidung des Europäischen Gerichtshofs (EuGH, Urteil vom 6. Oktober 2015, Az. C-362/14) eine vergleichsweise beliebte Methode, eine datenschutzrechtlich zulässige und sichere Datenübermittlung zu realisieren. Nach dem Urteil des EuGH kann jedoch der sichere Hafen nicht mehr als solcher eingestuft werden. Auch wenn diese Entscheidung gegenüber der irischen Datenschutzaufsichtsbehörde und mittelbar gegen Facebook ergangen ist, hat sie gleichwohl weitreichende Folgen für alle diejenigen, die personenbezogene Daten aus Europa in die USA übertragen.

Cloud-Speicherdienste wie zum Beispiel Dropbox, soziale Medien wie Facebook, oder auch sonstige IT-Dienstleister in den vereinigten Staaten – sie alle sind von der Ungültigkeit des Safe-Harbor-Abkommens betroffen. Es gibt aber natür-

lich auch Anbieter, die bereits reagiert haben, darunter beispielsweise Microsoft, das seine Office-365-Lösung mit der Beschränkung des Server-Standorts auf Europa anbietet.

IT-Sicherheit

Das Gesetz zur Erhöhung der Sicherheit informationstechnischer Systeme (IT-Sicherheitsgesetz) ist hierzulande am 25. Juli 2015 in Kraft getreten. Primär werden hierdurch Unternehmen aus kritischen Bereichen der Infrastruktur zur besseren Absicherung ihrer IT-Systeme verpflichtet.

Provider dürfen beispielsweise aus Gründen der Störungsvermeidung Verbindungsdaten ihrer Kunden für einen Zeitraum von maximal sechs Monaten speichern. Gleichzeitig betrifft eine ebenfalls eingeführte Neuregelung im Telemediengesetz (TMG), nämlich dessen § 13 Abs. 7, faktisch alle Anbieter von »geschäftsmäßig angebotenen Telemedien«, also jeden Betreiber einer nicht nur rein privaten Internetseite.

Diese müssen inzwischen die nach dem Stand der Technik möglichen Maßnahmen zur Absicherung ihrer IT-Systeme ergreifen, andernfalls riskieren sie ein Bußgeld und eventuell auch eine wettbewerbsrechtliche Abmahnung. Diese Pflicht gilt nach Maßgabe des Gesetzgebers jedenfalls so weit, wie dies technisch möglich und wirtschaftlich zumutbar ist. Welche Maßnahmen von wem bis wann zu realisieren sind, ist noch immer nicht so genau klar.

Ob jetzt jeder Webshop zum Beispiel eine verschlüsselte Verbindung ermöglichen muss, wie genau Hauptverbreitungswege von Schadsoftware eingedämmt werden können und ob das regelmäßige Einspielen von Updates, Sicherheitspatches et cetera ausreicht, muss die Zukunft zeigen. Wichtig ist sicherlich auch der Aspekt, dass im Rahmen von Verträgen mit Dritten, wie zum Beispiel Online-Werbepartnern, eine Festlegung entsprechender Pflichten erfolgen sollte.

Vorratsdatenspeicherung 2.0

Nachdem Deutschland vor Jahren bereits einmal versucht hat, die Vorratsdatenspeicherung einzuführen, ist 2015 sozusagen die Version 2.0 in Kraft getreten. Notwendig wurde diese erneute Gesetzesnovelle, weil das Bundesverfassungsgericht die ursprüngliche Regelung aus dem Jahre 2007 als unvereinbar mit dem grundrechtlich garantierten Post- und Fernmeldegeheimnis aus Art. 10 Grundgesetz (GG) und damit als verfassungswidrig eingestuft hatte.

Mit Urteil vom 8. April 2014 (Az. C-293/12 und C-594) hat dann der EuGH nachgezogen und die EG-Richtlinie über die Vorratsspeicherung von Daten, die bei der Bereitstellung öffentlich zugänglicher elektronischer Kommunikationsdienste oder öffentlicher Kommunikationsnetze erzeugt oder verarbeitet werden, für ungültig erklärt.

Nach einigen Korrekturen ist dann am 16. Oktober 2015 das Gesetz zur Neuregelung der Telekommunikationsüberwachung und anderer verdeckter Ermittlungsmaßnahmen abgesegnet worden. Hauptbestandteil dieser Novelle ist die Verpflichtung von Telekommunikationsanbietern zur Speicherung von sogenannten Verkehrsdaten für zehn Wochen, Standortdaten von Handys sind hingegen nur vier Wochen lang aufzubewahren. Ausnahmen gibt es ausschließlich für soziale und kirchliche Beratungsstellen, wie etwa die Telefonseelsorge. Daten von Rechtsanwälten, Ärzten oder auch Steuerberatern dürfen hingegen gespeichert werden.

Links zum Thema

- Datenbank mit den bereits für Safe Harbor registrierten Unternehmen
  https://safeharbor.export.gov/list.aspx
- Standardvertragsklauseln der EU-Kommission
  http://eur-lex.europa.eu/LexUriServ/LexUriServ.do?uri=OJ:L:2010:039:0005:0018:DE:PDF
- 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
- Video-Trainings
  www.video2brain.com/de/trainer/michael-rohrlich

Whistleblower

Gleichzeitig wurde ein neuer Straftatbestand eingeführt, nämlich die sogenannte Datenhehlerei. Normiert werden soll sie in § 202d des Strafgesetzbuches (StGB). Diese Norm hat mit der VDS an sich nichts zu tun, sie stellt wohl eher einen Anti-Whistleblower-Paragrafen dar.

Mit Freiheitsstrafe bis zu fünf Jahren oder mit Geldstrafe kann derjenige bestraft werden, der Daten, die ein anderer ausgespäht oder sonst durch eine rechtswidrige Tat erlangt hat, sich oder einem anderen verschafft, einem anderen überlässt, verbreitet oder sonst zugänglich macht, um sich oder einen Dritten zu bereichern oder einen anderen zu schädigen. Daten sind hierbei nur solche, die elektronisch, magnetisch oder sonst nicht unmittelbar wahrnehmbar gespeichert sind oder übermittelt werden (§ 202a Abs. 2 StGB).

Diese neue Strafnorm ist spätestens den Veröffentlichungen von Edward Snowden in der Diskussion. Pikant an der Neuregelung ist, dass für die Presse zwar eine Ausnahme vorgesehen ist. Diese Privilegierung gilt jedoch nur für »berufsmäßig handelnde Journalisten«. Somit dürften also beispielsweise ehrenamtlich tätige Blogger et cetera nicht erfasst sein. ■

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

Arbeiten in der Großstadt

Digitalisierung und Industrie 4.0 sind die Schlagworte für die Anpassungen, welche die deutschen Unternehmen gegenwärtig vornehmen. Nur wenige warten noch ab, viele stellen gerade die Agenda dafür auf und manche haben bereits mit der Umsetzung begonnen – und alle brauchen dafür Entwickler. Folglich sagen alle Prognosen voraus, dass der Mangel an Software-Entwicklern auch in den nächsten Jahren anhalten wird. Im Umkehrschluss bedeutet das für alle, die heute schon als Entwickler arbeiten, dass sie sich ihren Arbeitgeber aussuchen können. Bis zu einem gewissen Grad gilt das auch für den Arbeitsort. Allerdings werden mehr als die Hälfte der Angebote für die 15 größten deutschen Städte ausgeschrieben.

Wer einen Job als Web-entwickler in einer der Big-15-Städte sucht, kann diesen auch nach der neuesten Auswertung am leichtesten in München (3091 Treffer), Berlin (252 Treffer) oder Hamburg (2134 Treffer) finden (Bild 1). Allerdings belegt Berlin wieder mit deutlichem Abstand den zweiten Platz, der im Vormonat ganz knapp an Hamburg ging. Über 1000 Treffer lieferte die Abfrage zudem noch für Frankfurt, Düsseldorf und Stuttgart. Köln liegt auf dem siebten Rang. Insgesamt sind laut der aktuellen Auswertung rund 66 Prozent der Jobs für Webentwickler in den Big-15-Städten zu finden.

In Bayern, Baden-Württemberg, NRW und Berlin gibt es derzeit die meisten Jobangebote für Mobile-Entwickler (Bild 2). Die durchgeführte Auswertung nach Städten zeigte, dass in München, Berlin, Hamburg und Frankfurt die besten Adressen für arbeitsuchende App-Entwickler sind. Mit 64,4 Prozent liegt der Anteil der Jobs in Großstädten ähnlich hoch wie bei den Webentwicklern.

Die Abfrage der in Angeboten nachgefragten Technologiekenntnisse zeigte einen leichten Rückgang der Gesamtzahl der Treffer um 1,4 Prozent. Gegen den allgemeinen Trend verstärkte sich die Nachfrage nach Cloud- sowie nach iOS-Kenntnissen. Tabelle 1 zeigt den aktuellen Stand der Nachfrage nach Technologiekenntnissen.

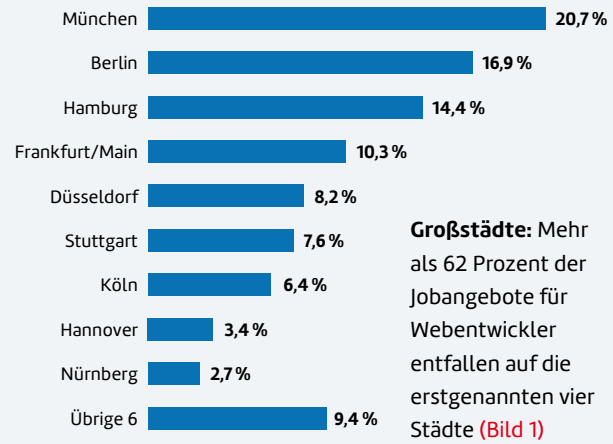
Die in Stellenangeboten mit großem Abstand am häufigsten genannte Programmiersprache ist weiterhin Java. Sowohl in der Datenbank von Jobkralle.de als auch in der von StepStone lag Java am Erhebungsstichtag mit großem Vorsprung vorne. Auf dem zweiten Platz folgt bei Jobkralle die Skriptsprache PHP, während bei StepStone C++ am zweithäufigsten nachgefragt wurde. Bei Jobkralle kam C++ nur auf dem vierten Rang. HTML, JavaScript und C# folgen bei StepStone auf den Plätzen drei bis fünf und bei Jobkralle schafften es ebenfalls C# und C++ sowie die SAP-Datenbanksprache ABAP in die Top 5.

Tabelle 1

Rang	Technologie	Anteil *
1	Cloud	14,5 %
2	MySQL	11,3 %
3	SharePoint	9,6 %
4	HTML5	9,2 %
5	Responsive Web	8,2 %
6	Big Data	6,4 %
7	Microsoft SQL Server	6,0 %
8	Android	6,0 %
9	Windows 10	5,9 %
10	iOS	5,6 %
11	CSS3	4,4 %
12	Angular.js	3,9 %
13	ASP .NET	2,9 %
14	WPF	2,4 %
15	NoSQL	2,2 %
16	WCF	1,5 %

* Prozentualer Anteil der Treffer

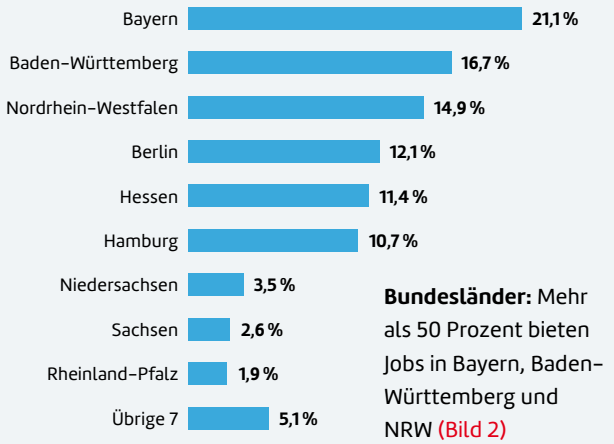
Jobs für Webentwickler



Großstädte: Mehr als 62 Prozent der Jobangebote für Webentwickler entfallen auf die erstgenannten vier Städte (Bild 1)

web & mobile developer 1/2016

Jobs für Mobile-Entwickler



Bundesländer: Mehr als 50 Prozent bieten Jobs in Bayern, Baden-Württemberg und NRW (Bild 2)

web & mobile developer 1/2016

Zahl des Monats

In Deutschland arbeiten inzwischen **1.002.000** Menschen in ITK-Unternehmen (IT, Telekommunikation und Unterhaltungselektronik). In den vergangenen fünf Jahren sind rund **135.000** neue Arbeitsplätze entstanden. Allein im Jahr 2015 kamen fast **25.000** neue Jobs dazu.

Quelle: Bitkom.org

Bitkom

Software als Umsatzlokomotive

Gemäß aktueller Bitkom-Zahlen hat die ITK-Branche ihre Position als zweitgrößter Arbeitgeber gefestigt. Das Wachstum an Arbeitsplätzen findet dabei ausschließlich in den IT-Unternehmen statt, auf die 794.000 Beschäftigte entfallen. Der Umsatz der Branche ist 2015 laut vorläufigen Zahlen auf 156 Milliarden Euro angewachsen – das sind 1,9 Prozent mehr als im Vorjahr. Wachstumstreiber bleibt die Informationstechnologie. Die Umsätze sollen im Jahr 2015 um 3,5 Prozent auf 80,4 Milliarden Euro angestiegen sein, wobei der Bereich Software mit einem Plus von 5,4 Prozent auf 20,1 Milliarden Euro das stärkste Wachstum verzeichnet (Bild 3).

Über die vergangenen Jahre kontinuierlich zugelegt hat das Geschäft mit IT-Dienstleistungen wie IT-Beratung und das Projektgeschäft, welches 2015 um 3,0 Prozent auf 37,3 Milliarden Euro gewachsen ist. »Die Anbieter profitieren davon, dass Unternehmen aller Branchen ihr Geschäft auf die Digitalisierung ausrichten«, so Bitkom-Chef Rohleder. Überraschend positiv entwickelt sich in diesem Jahr zudem der Umsatz mit IT-Hardware, der um 2,8 Prozent auf 23,0 Milliarden Euro wachsen soll.

Bitkom

Prognose für 2016

Für 2016 soll der ITK-Gesamtmarkt weiter wachsen. Der Branchenverband Bitkom erwartet ein Umsatzwachstum um 1,5 Prozent auf 158,4 Milliarden Euro. Neben einem voraussichtlich schwächeren Geschäft mit PCs und Sättigungseffekten bei Endgeräten der Telekommunikation sorge vor allem der VW-Effekt für Unsicherheit bei IT-Dienstleistern und Software-Anbietern. »Nicht nur die Automobilhersteller der Volkswagen-Gruppe sind zurückhaltend, die Unsicherheit setzt sich über Wettbewerber und Zulieferer fort und endet bei Städten und Gemeinden, die stark auf Steuereinnahmen der Automobilindustrie angewiesen sind«, so Rohleder. Eine detaillierte Marktprognose will der Verband im Vorfeld der CeBIT 2016 vorstellen.

StepStone

Jobbörse in den Top 5

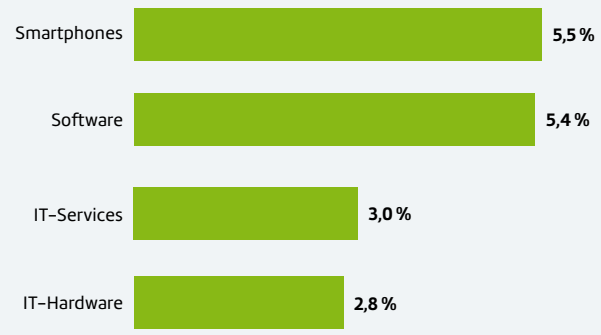
Was haben Wikipedia, Google, WhatsApp, Skype und StepStone gemeinsam? Sie gehören zu den fünf beliebtesten Online-Marken der Deutschen! In der von amerikanischen Unternehmen dominierten Online-Branche ist StepStone damit das einzige Unternehmen mit Hauptsitz in Deutschland unter den Top 5. Das ist das Ergebnis einer repräsentativen Befragung von 700.000 Verbrauchern durch das Handelsblatt und das Marktforschungsinstitut YouGov. Die deutsche Online-Jobbörse StepStone belegt in der Rangfolge der beliebtesten Internetmarken Platz 5.

IT-suppliers.com

Mobil in die Zukunft

Die Zeichen für die Mobile Economy stehen weiter auf Wachstum. Nach einer Umfrage von IT-suppliers.com glauben nahezu drei Viertel der befragten Unternehmen, dass Internet-Start-ups, die sich auf Lösungen und Geschäftsmodelle für mobile Endgeräte spezialisieren, die besten Zukunftsaussichten besitzen. Auf den weiteren Plätzen liegen Cloud- und Big-Data-Lösungen (Bild 4). Interessant dabei: Die aktuelle Auftragslage bildet die positive Zukunftserwartung derzeit noch nicht ab. Denn bei der Frage, welche Projekte derzeit am meisten extern vergeben werden, antworten die IT-Dienstleister, dass es sich in erster Linie um die Erstellung von Webseiten, ESB-Projekte (Enterprise Service Bus) sowie Beratungsaufträge handelt.

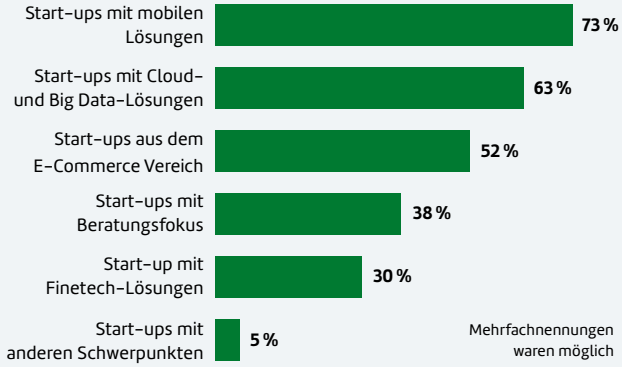
Wachstum 2015



Umsatzentwicklung ausgewählter IT-Bereiche im Jahr 2015 im Vergleich zum Vorjahr (Bild 3).

web & mobile developer 1/2016 Quelle: Bitkom

Start-up-Chancen

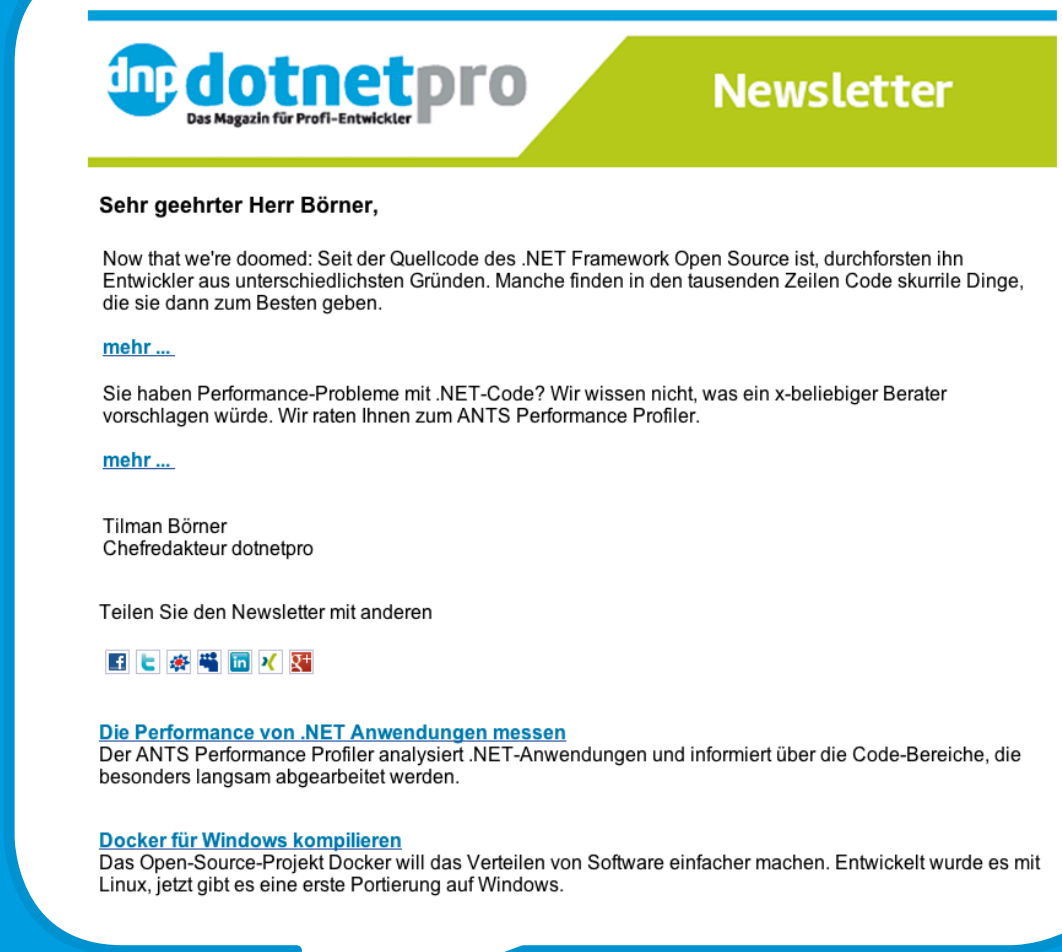


Antworten: Welche Start-up-Unternehmen werden in den nächsten drei Jahren am erfolgreichsten sein? (Bild 4)

web & mobile developer 1/2016 Quelle: IT-suppliers.com

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

5Minds IT-Solutions GmbH & Co. KG
Pfefferackerstraße 5
45894 Gelsenkirchen
T: +49 (0) 209/ 883 068 – 0
F: +49 (0) 209/883 068 – 99
Nicole.Jones@5Minds.de
www.5Minds.de

Wir entwickeln hochperformante Enterprise-Lösungen für Desktop, Web und Mobile mit aktuellen Technologien (u.a. .NET, Xamarin, Phonegap, Node.js, SQL, NoSQL, Messaging). Dabei verstehen wir uns nicht nur als Problemlöser, sondern als Vordenker für Softwareentwicklungsprozesse. Wo andere bei der Beratung enden, gehen wir noch einen Schritt weiter. Wir blicken für Sie über den Tellerrand und entwickeln nachhaltige und maßgeschneiderte IT-Lösungen, damit Sie zukünftigen Anforderungen gelassen entgegenblicken können.
Wir freuen uns auf Sie! Nicole Jones

ANEXIA Internetdienstleistungs 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

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.
Seit über 15 Jahren vertrauen uns deshalb Marktführer wie Aldi Süd, Microsoft und Siemens.
prodot – People. Passion. Performance..

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

Die Ausgabe 2/2016 erscheint am 14. Januar 2016

Zahlungssysteme in E-Commerce-Applikationen integrieren

Der kommerzielle Erfolg eines Webshops mag von einer Vielzahl verschiedener Faktoren abhängen, doch keine einzige Bestimmungsgröße wirkt sich derart nachhaltig auf die Konversionsrate aus wie der Ablauf des Zahlungsvorgangs aus der Sicht des potenziellen Käufers. Dabei stellt sich heraus, dass weder Suchmaschinenoptimierung der einzelnen Produktseiten, noch A/B- oder MV-Tests verschiedener Designvarianten zur Maximierung der Konversionsrate, noch der Einsatz eines CDNs (Content Delivery Networks) zur Minimierung der Ladezeiten einem Webshop zum Erfolg verhelfen können, wenn der Besucher das unterstützte Bezahlsystem grundlegend ablehnt. Daher sind Online-Zahlungssysteme mit einer breiten Akzeptanz für ein Unternehmen nahezu überlebenswichtig.

Regressionstest mit WebDriverIO

Eine Software- oder Webanwendung erfährt in der Regel auch nach ihrer Veröffentlichung mehrere Änderungen, sei es durch Hinzufügen neuer Features oder Beheben von Fehlern. Hierbei kann es leicht passieren, dass diese Änderungen ungewollte Auswirkungen auf andere bereits bestehende Komponenten der Anwendung haben beziehungsweise dort zu neuen Fehlern führen. Das ist der Punkt, an dem Regressionstests ansetzen.

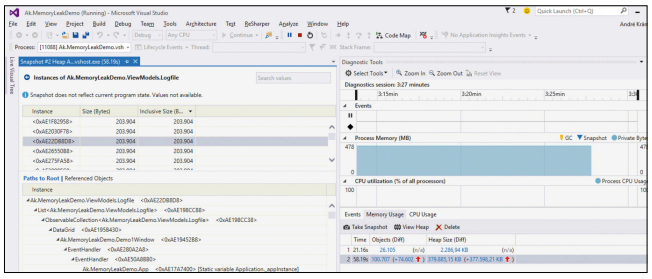
Digitale Transformation

Die digitale Transformation stellt Unternehmen vor vielfältige Herausforderungen. Eine davon ist das Thema Enterprise Mobility. In der Geschäftswelt der Zukunft nutzen Mitarbeiter stationäre Desktop-PCs und Notebooks nur noch im Notfall. Stattdessen verwenden sie mobile Devices: Smartphones, Tablets und Wearables wie etwa Smartwatches und Smartglasses. Über diese greifen sie auf diverse mobile Applikationen zu.

Datenanalyse mit BI-Tools

Noch vor wenigen Jahren war die Einführung von Business Intelligence eine unternehmensweite strategische Entscheidung, von langer Hand geplant, sorgfältig vorbereitet und von den Experten der IT-Abteilung umgesetzt. Analysewerkzeuge und Reporting-Tools sollten vor allem zur Konsolidierung und Standardisierung der Unternehmensprozesse beitragen. Heute sind es die Fachabteilungen, die die Einführung neuer BI-Tools forcieren.

dotnetpro



Ausgabe 1/2016 ab 17.12.2015 am Kiosk

Im nächsten Schwerpunkt der dotnetpro geht es ans Eingemachte: Dinge, die tief drinnen im .NET Framework ablaufen, aber massive Auswirkungen auf die Anwendungen haben. Da geht es zum Beispiel um Memory Leaks, Namensraum System und Runtime.Caching.
www.dotnetpro.de

Unsere digitalen Angebote

Wöchentlicher Newsletter
www.webundmobile.de/newsletter-1022034.html

Shop
https://shop.webundmobile.de

YouTube
www.youtube.com/user/developermedia

Facebook
www.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

Wir machen Sie digitalmobil.

Wir liefern passgenaue Strategien und Lösungen für Ihre Inhalte auf

- iPhone/iPad
- Android
- BlackBerry
- Windows Phone 7
- dem mobilen Browser

digitalmobil

Besuchen Sie uns unter www.digitalmobil.com