

Sechs MVC-Frameworks im Vergleich

Zur Ansicht

Golo Roden

Der JavaScript-Markt bietet zahlreiche Frameworks zur Entwicklung clientseitiger Anwendungen. Alle dienen zwar demselben Zweck, verfolgen jedoch unterschiedliche Philosophien. Ein bei Programmierern beliebter Schwerpunkt ist die Unterstützung des MVC-Paradigmas. Aber auch da gibt es deutliche Unterschiede.

Entwurfsmuster – ein ursprünglich aus der Architektur stammender Begriff – stellen Softwareentwicklern wiederverwendbare Vorlagen zur Lösung spezifischer Aufgabenstellungen zur Verfügung. Ein für Single-Page-Anwendungen häufig genutztes ist das Model-View-Controller-Muster (MVC). Es stammt bereits aus dem Jahr 1979 und wurde erstmals von Trygve Reenskaug beschrieben, der damals an der Entwicklung der Sprache Smalltalk im Xerox Palo Alto Research Center arbeitete. Die Idee hinter MVC ist die Unterteilung einer Anwendung in drei Komponenten, die aufeinander aufbauen: das Model, den View und den Controller. Es dient zahlreichen Anwendungen als Basis und bildet außerdem das Fundament für das moderne Entwurfsmuster Model-View-ViewModel (MVVM).

Der View ist in beiden Mustern gleich definiert: Es handelt sich dabei um die tatsächliche Anzeige der Anwendung, einschließlich der Steuerelemente und der Anzeigelogik wie Animationen oder anderer Effekte. All diesen Effekten ist gemein, dass sie nichts zur eigentlichen Funktion der Anwendung beitragen, also nicht geschäftskritisch sind.

Ebenfalls in beiden Mustern gleich definiert ist das Model: Es implementiert die fachliche Domäne, das heißt, es stellt die geschäftsrelevanten Funktionen und Daten zur Verfügung und verarbeitet sie. Das Model ist allerdings kein reines Datenbankmodell, kann ein solches aber enthalten – es kann sich aber auch rein im Speicher befinden.

Kleiner Unterschied mit großer Wirkung

Der Unterschied zwischen den beiden Ansätzen MVC und MVVM liegt darin, wie die Modelle jeweils View und Model verbinden. In MVC übernimmt der Controller diese Aufgabe (siehe Abbildung 1), der Eingaben aus dem View liest, Funktionen im Model aufruft, einen neuen View generiert und anschließend Daten darin ausgibt. Dieses Muster findet sich häufig in Web-Frameworks wie ASP.NET MVC, Ruby on Rails oder Sails.js. Der View entspricht dabei dem HTML-Code, der Controller dem clientseitigen JavaScript- respektive dem serverseitigen proprietären Code. Das Model ist die fachliche Implementierung, die der Controller anspricht.

Als größter Nachteil von MVC gilt die enge Bindung zwischen View und Controller: Beide unabhängig voneinan-

der zu entwickeln – beispielsweise von Designern und Entwicklern –, fällt häufig schwer, da sich Änderungen an der einen Komponente direkt auf die jeweils andere auswirken.

Dieses Problem versucht das MVVM-Muster zu lösen, indem es auf einen Controller verzichtet und ihn durch ein sogenanntes ViewModel ersetzt. Dabei handelt es sich um die Abbildung des View im Programmcode. Gehört beispielsweise eine Textbox zum View, enthält das ViewModel eine Eigenschaft vom Typ *string*. Enthält der View hingegen eine Schaltfläche, beinhaltet das ViewModel eine aufrufbare Funktion.

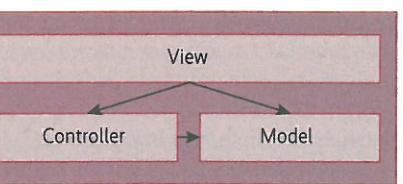
Die Bindung zwischen beiden stellt der Webentwickler über (zumeist bidirektionale) Datenbindung her, sodass er hierfür nichts von Hand programmieren muss (siehe Abbildung 2). Dadurch eignet sich MVVM besser für die Zusammenarbeit mit Designern, da die Datenbindung für sie quasi „unsichtbar“ ist. Außerdem lässt sich das ViewModel leicht testen, was be-

deutet, dass UI-Tests einfach und mit den üblichen Mitteln durchführbar sind (siehe Abbildung 3).

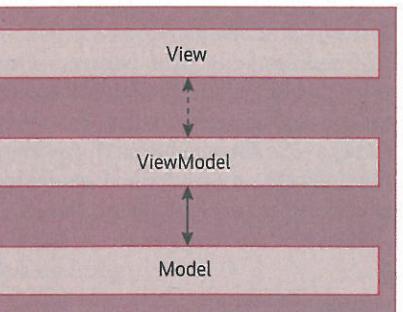
Im Standard ist vieles noch Zukunftsmusik

Die zukünftigen Versionen von JavaScript, deren nächste – ECMAScript 6 – für Juni 2015 angekündigt ist, sollen zahlreiche neue Sprachmerkmale enthalten, die unter anderem auch das MVC-Muster unterstützen. Für einige geplante Funktionen steht bereits heute eine Implementierung als Drittmodul zur Verfügung. Eine native Unterstützung verspricht jedoch häufig nicht nur eine vereinfachte Verwendung, sondern zugleich eine beschleunigte Ausführung.

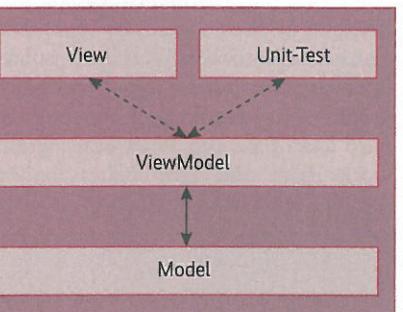
Ein gutes Beispiel ist die – allerdings erst für ECMAScript 7 vorgesehene – Funktion *Object.observe()*: Sie erlaubt das einfache Überwachen eines Objekts auf Änderungen und bildet so die Basis für



Bei MVC besteht eine enge Bindung zwischen View und Controller (Abb. 1).



Das MVVM-Muster setzt das ViewModel anstelle des Controllers und beseitigt so die enge Datenbindung (Abb. 2).



Da die Datenbindung bei MVVM quasi unsichtbar ist, lässt sich das ViewModel leicht mit den üblichen Mitteln testen (Abb. 3).

X-TRACT

- Frei verfügbare Frameworks ergänzen JavaScript um Funktionen, die erst für zukünftige Versionen des ECMAScript-Standards geplant sind.
- Einige dieser Frameworks setzen den Schwerpunkt auf das MVC-Entwurfsmuster und unterstützen so den Webentwickler bei der klaren Strukturierung seines Codes.
- Trotz vieler Gemeinsamkeiten unterscheiden sich die sechs in diesem Artikel vorgestellten Frameworks hinsichtlich der Konsequenz, mit der sie den Ansatz „Model, View, Controller“ verfolgen.

JAZOON'14

21-22 OCTOBER 2014, ZÜRICH



Keynote Speaker
Mike Milinkovich
(Eclipse Foundation)



Keynote Speaker
Gunter Dueck

Find more information and get your ticket now at: www.jazoon.com

iegliche Art von Datenbindung. Selbstverständlich lässt sich ein derartiges Verhalten bereits heute umsetzen, jedoch ist der hierfür erforderliche Aufwand ohne zusätzliche Bibliotheken immens.

Denn allen Neuerungen zum Trotz benötigt die Entwicklung einer komplexen Anwendung mehr als lediglich eine Programmiersprache mit tragfähigen Konzepten: Zum einen bedarf die Struktur des zu schreibenden Codes besonderer Aufmerksamkeit, zum anderen unterstützt das Einhalten etablierter Entwurfsmuster eine zügige und qualitativ anspruchsvolle Entwicklung. All das leisten Frameworks, die auf die Entwicklung clientseitiger Anwendungen spezialisiert sind.

Frameworks bieten ein Komplettpaket

Das impliziert allerdings, dass die Wahl eines Frameworks nicht nur eine kurzfristige Entscheidung ist: Betrachtet man als den eigentlichen Wert einer solchen Software die Abstraktion über die zugrunde liegende Programmiersprache, wird klar, dass die Wahl eines Frameworks über eine rein technische Entscheidung weit hinausgeht und die Architektur und Struktur einer Anwendung grundlegend prägt.

Listing 1: Knockouts ViewModel-Ansatz

```
<input type="text" data-bind="value: message">
<button data-bind="click: send">Senden</button>

<script>
var ViewModel = function () {
    this.message = ko.observable();
    this.send = function () {
        alert(this.message());
    };
}
ko.applyBindings(new ViewModel());
</script>
```

Listing 2: React.renderComponent() mit vorgegebenen Standardwerten

```
var HelloWorld = React.createClass({
getInitialState: function () {
    return {
        message: 'Hello, world!'
    };
},
render: function () {
    return React.DOM.div({}, 
        new React.DOM.input({
            placeholder: this.props.defaultText,
            value: this.state.message
        }));
},
React.renderComponent(
    new HelloWorld({
        defaultText: 'Enter your message here...'
    }),
    document.body
);
```

Anhand sechs ausgewählter Frameworks zeigt dieser Artikel, welche Funktionen den jeweiligen Programmierer bei seiner Arbeit speziell unterstützen und für welche Anwendungsfälle sie besonders geeignet sind.

Knockout.js

Das einfachste der in diesem Artikel vorgestellten Frameworks ist Knockout.js. (Zu Bezugsquellen und weiterführenden Informationen siehe „Onlinequellen“ und „Alle Links“ sowie die Tabelle auf S. 52.) Diese Bewertung bezieht sich gleichermaßen auf seine Verwendbarkeit und seinen Funktionsumfang: Die einzige Aufgabe des Frameworks besteht darin, das Entwurfsmuster Model View ViewModel (MVVM) für HTML zu implementieren und auf diesem Weg das Konzept der Datenbindung zwischen HTML und JavaScript zu ermöglichen.

Die Idee hinter MVVM ist, zwischen Ansicht (View) und Domänenmodell (Model) eine weitere Schicht einzufügen, die eine lose Kopplung der Ansicht ermöglicht. Da diese Schicht in ihrem Aufbau dem konkreten View entspricht, bezeichnet man sie als ViewModel. Sie enthält für jedes UI-Element passende Daten, die Knockout über Datenbindung an die jeweilige Ansicht koppelt. Verwendet man hierfür eine bidirektionale Datenbindung, wirken sich Eingaben des Anwenders direkt auf das ViewModel und Änderungen im Quellcode am ViewModel direkt auf die Ansicht aus. Zugleich lösen Schaltflächen die gewünschten Funktionen am ViewModel aus.

Ein wesentlicher Vorteil dieses Vorgehens besteht darin, sämtliche UI-Logik aus der Ansicht in das zugehörige ViewModel zu verlagern, das der Entwickler auf einfacherem Weg automatisiert testen kann. Umfangreiche, fehleranfällige und vor allem aufwendige, von Hand durchzuführende UI-Tests können daher entfallen.

All das meistert Knockout mit Bravour. Es bedarf nur weniger Zeilen JavaScript-Code und einiger zusätzlicher Attribute im HTML-Markup, um ein ViewModel zu definieren und die Datenbindung zu aktivieren (siehe Listing 1).

Der Teufel liegt allerdings, wie so häufig, im Detail. Mehr als einen bequemen Weg zur Datenbindung bietet Knockout nämlich nicht. Zum einen fehlt eine integrierte Möglichkeit, mit dem Server zu kommunizieren, obwohl das für Single-Page-Anwendungen sicherlich zu den grundlegenden Aufgaben gehört. Zum an-

deren, und dieser Mangel wiegt deutlich schwerer, erhält der Entwickler keine Hilfe bei der Verwaltung verschiedener Ansichten und dem Wechsel zwischen ihnen. Auch bei der Verarbeitung von Routen versagt Knockout die Unterstützung.

Letztlich ist das System für all diese Aufgaben auf Module anderer Frameworks angewiesen, etwa jQuery. Der Nachteil bei diesem Vorgehen liegt in den zusätzlichen Abhängigkeiten, die man eingeht und die sich unter anderem im Hinblick auf die langfristige Kompatibilität negativ auswirken könnten. Zudem steigen der Wartungsaufwand und das Risiko von Inkompatibilitäten zwischen den zusätzlichen Modulen.

Trotz allem ist Knockout einen Blick wert – vor allem, wenn eine Anwendung nur aus wenigen Ansichten besteht und Funktionen zu deren Routing und Verwaltung nicht benötigt.

React

Komplexe Ansichten, die viele Steuerelemente enthalten, fordern MVC-Frameworks in besonderem Maße. Da diese Frameworks für jedes Steuerelement Datenbindung vorsehen, steigt der Verwaltungsaufwand mit der Komplexität der Ansichten. Das verlangsamt Anwendungen und deren Antwortverhalten (Reaktivität) deutlich.

Ahilfe verspricht ein verhältnismäßig junges Framework namens React, das von Facebook entwickelt wird. Desse grundlegende Idee ist, ein virtuelles DOM (Document Object Model) im Arbeitsspeicher zu verwenden und es nur bei Bedarf mit dem realen DOM des Webbrowsers abzugleichen. Dadurch lassen sich Änderungen am (virtuellen) DOM ausgesprochen schnell durchführen, insbesondere für komplexe Ansichten.

Anders als Knockout fordert React vom Entwickler keine Anpassung des HTML-Codes. Sämtliche UI-Programmierung erfolgt vollständig in JavaScript. Zum Erzeugen von Komponenten stehen verschiedene Funktionen wie *React.DOM.div()* und *React.DOM.span()* zur Verfügung. Sie lassen sich zudem verschachteln, um komplexere Komponenten zusammenzusetzen. Ein Aufruf der Funktion *React.renderComponent()*

fügt die gewünschten Komponenten schließlich dem DOM des Webbrowsers hinzu:

```
React.renderComponent(
    new React.DOM.div({}, 'Hello, world!'),
    document.body
);
```

Der Definition benutzerdefinierter Komponenten dient die Funktion *React.createClass()*. Sie erwartet in einem Optionsobjekt zumindest eine *render()*-Funktion, die das Zeichnen der Komponente übernimmt. Zusätzlich lassen sich Funktionen implementieren, die den Status einer Komponente und damit letztlich ihre Darstellung beeinflussen. Beim Aufruf von *React.renderComponent()* kann der Entwickler darüber hinaus Standardwerte für die Komponente vorgeben (siehe Listing 2).

Auf einem ähnlichen Weg kann React auf Veränderungen reagieren, die der Anwender im Webbrowser auslöst, beispielsweise Tastatur- und Mauseingaben. Außerdem können Komponenten einander referenzieren, um sich gegenseitig zu beeinflussen und zu modifizieren. So lassen sich aus einigen wenigen Grundbausteinen durchaus komplexe Ansichten entwickeln, die dennoch eine gute Leistung aufweisen.

Bemerkenswert an React ist, dass es sich um kein vollständiges MVC-Framework handelt, sondern lediglich um ein optimiertes Modul zur leistungsfähigen Darstellung von Ansichten. Daher lässt es

sich zur Leistungsoptimierung in andere Frameworks wie Backbone und AngularJS integrieren.

React ist sicherlich kein JavaScript-Modul, das unbedingt jeder UI-Entwickler in seinem Portfolio haben sollte. Ab einer gewissen Komplexität der Ansichten einer Anwendung ist ein Blick auf React aber durchaus ratsam.

Backbone.js

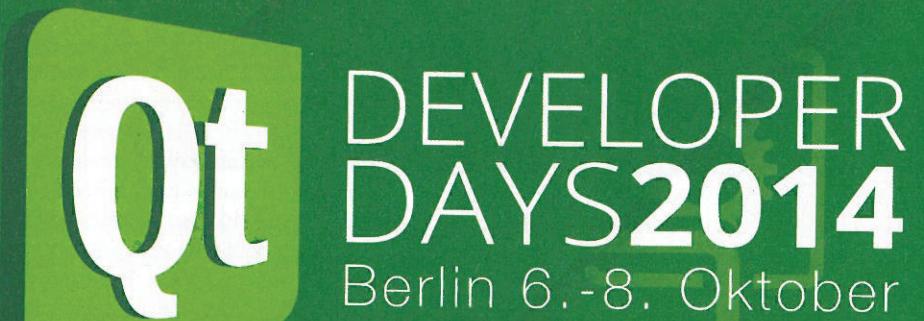
Zu den ältesten und historisch bedingt am weitesten verbreiteten MVC-Frameworks für JavaScript zählt Backbone.js. Mit React teilt es den Ansatz, JavaScript getrieben vorzugehen. Anders als React (und Knockout) enthält Backbone allerdings kein integriertes Konzept zur Darstellung von Ansichten. Stattdessen konzentriert sich das Framework auf die Modellierung von Entitäten und Entitätslisten sowie deren Synchronisation mit einem Webdienst.

Den Kern des Frameworks bildet die Klasse *Backbone.Events*, von der sich wiederum Klassen wie *Backbone.Model* und *Backbone.Collection* ableiten. Da all

diese Klassen das Beobachtermuster implementieren, ermöglicht das eine entkoppelte Kommunikation innerhalb einer Backbone-Anwendung. Das Framework schreibt dem Entwickler keine spezielle Architektur vor, bietet ihm aber eine Reihe von Hilfsmitteln zur Implementierung einer sauberen Struktur.

Für die Modellierung einer Domäne stehen die beiden zuvor genannten Klassen *Backbone.Model* sowie *Backbone.Collection* zur Verfügung. Erstere beschreibt einzelne Entitäten, letztere eine Liste von diesen. Die beiden Klassen bilden daher eine 1:n-Beziehung. Im Gegensatz zu vielen anderen Frameworks können Entitäten und Listen bei Bedarf nicht nur Daten, sondern auch Geschäftslogik enthalten.

Backbone implementiert die Persistenz eines derartigen Domänenmodells über eine integrierte Synchronisation mit einem vom Entwickler definierten REST-basierten Webdienst (Representational State Transfer). Entitäten und Listen enthalten von Haus aus spezielle Funktionen zum Laden, Speichern und Löschen. Im Hintergrund verwendet Backbone für den Datenabgleich mit dem Webserver



Finden Sie heraus, warum Qt die schlauste Wahl ist:

Plattformunabhängig
Flexibel
Kürzere Entwicklungszeit

Qt Developer Days Europe, Berlin, 6.-8. Oktober
Das herausragende Qt-Ereignis des Jahres
Jetzt anmelden: www.qtdeveloperdays.com/europe #QtDD14



JavaScript-Frameworks mit MVC- oder MVVM-Konzept

Framework	Knockout.js	React	Backbone.js	Ampersand.js	Ember.js	AngularJS
Konzept	bidirektionale Datenbindung für HTML	virtuelles DOM für schnelles Rendering	Entitäten und Listen als Basis für Views	Entitäten und Listen als Basis für Views, modulare Struktur	Single-Page-Anwendungen in JavaScript	HTML für Single-Page-Anwendungen
Entwickler	Steve Sanderson	Facebook	Jeremy Ashkenas	&yet	Yehuda Katz und Tom Dale	Google
URL	knockoutjs.com	facebook.github.io/react/	backbonejs.org	ampersandjs.com	emberjs.com	angularjs.org
Version	v3.2.0	v0.11.1	v1.1.2	verschiedene Versionen für Module	v1.7.0	v1.2.23
Typ	MVVM	V	MVC	MVC	MVC / MVVM	MVC / MVVM
Lizenz	MIT	Apache 2	MIT	MIT	MIT	MIT
Größe (minifiziert)	57 KByte	545 KByte	20 KByte	individuell	352 KByte	111 KByte
Abhängigkeiten	keine	keine	Underscore.js	Underscore.js	jQuery, Handlebars	keine
Deklarative Datenbindung	✓	-	-	-	✓	✓
Erweiterbares HTML	-	-	-	-	-	✓
Vorgegebene Komponententypen	-	-	✓	✓	✓	✓
Dependency Injection	-	-	-	-	✓	✓
Integrierte REST-Unterstützung	-	-	✓	✓	✓	✓
Routing und mehrere Ansichten	-	-	✓	✓	✓	✓
Webbrowser-Werkzeuge	-	-	-	-	✓	✓

eine interne Funktion, die sich bei Bedarf allerdings durch eine eigene Implementierung ersetzen lässt.

Zusätzlich kennt das Framework das Konzept von Views, sie entsprechen aber eher dem des Controllers als dem der Ansicht im MVC-Entwurfsmuster. Die tatsächliche Darstellung der Daten und deren Aktualisierung bleibt vollständig dem Entwickler überlassen.

Dadurch entstehen Backbone zwei Nachteile: Der Entwickler ist auf die Integration einer Bibliothek wie jQuery angewiesen, was langfristig zu Kompatibilitäts- und Wartungsproblemen führen kann. Zudem gibt er die saubere Struktur einer Backbone-Anwendung zumindest stellenweise zugunsten von Code auf, den er mit einer anderen Bibliothek entwickelt hat. Das kann sich insbesondere bei der Verwendung von jQuery schnell rächen.

Darüber hinaus übernimmt Backbone von sich aus keinerlei Verwaltung der Ansichten. Das bedeutet, dass der Entwickler selbst Sorge für deren korrekte Entsorgung tragen muss. Allzu rasch entstehen hierbei jedoch nur schwer auffindbare Speicherlecks, die der Stabilität und Leistung der Anwendung langfristig schaden.

Unterm Strich ist Backbone ein zwar leistungsfähiges, aber durchaus auch in die Jahre gekommenes Framework. Im Vergleich zu moderneren Alternativen wirken zahlreiche Design-Entscheidungen innerhalb des Systems nicht mehr zeitgemäß und verursachen unnötig viel

Aufwand für den Entwickler. Außerdem führt Backbone rasch zu einer übertriebenen Struktur des Codes, was dessen Verständnis deutlich erschwert.

Ampersand.js

Das von der Firma &yet stammende Framework Ampersand.js entspricht weitgehend einer überarbeiteten Fassung von Backbone. Es löst einige der grundlegenden Probleme von Backbone und glänzt insbesondere durch eine mögliche Modularisierung. An die Stelle eines Monolithen tritt eine Sammlung einzelner, eigenständiger Komponenten, die sich über npm, den Package Manager von Node.js, installieren und über dessen Tool browserify integrieren lassen.

Das bedeutet, dass sich Anwendungen mit Ampersand im Stil von Node.js-Modulen schreiben lassen. Zudem fällt es deutlich leichter, angepasste Versionen von Ampersand zu bauen oder einzelne Module durch benutzerdefinierte Varianten auszutauschen. Im Gegensatz zu Backbone unterstützt Ampersand über das Modul *ampersand-view* auch hierarchisch verschachtelte Ansichten, was die Entwicklung komplexerer Ansichten fördert.

Außerdem steht dem Entwickler das Modul *ampersand-view-switcher* zur Verfügung, mit dem er Ansichten wechseln kann, ohne sich manuell um das Aufräumen der nicht mehr benötigten Views kümmern zu müssen. Das verringert das

Risiko für Speicherlecks deutlich und beugt daher einen der größten Stolpersteine von Backbone.

Positiv fällt ebenfalls das integrierte Kommandozeilenwerkzeug auf, das das Grundgerüst für vollständige Anwendungen sowie einzelne Komponenten generieren kann. Das fördert nicht nur die Standardisierung des Codes, sondern unterstützt außerdem den Einsatz bewährter Entwurfsmuster. Insbesondere die Möglichkeit, aus einem gegebenen Model eine Ansicht generieren zu lassen, erweist sich im Alltag als hilfreich.

Auf der Webseite tools.ampersandjs.com steht zudem eine Liste handverlesener npm-Module zur Auswahl, die das Framework um zusätzliche Funktionen ergänzen können. Zumindest derzeit stammt der Großteil dieser Module allerdings von &yet selbst. Es bleibt also abzuwarten, ob sich wie bei den anderen Frameworks auch um Ampersand eine Community sammeln wird, die das Projekt langfristig voranbringen kann.

Da Ampersand prinzipiell dieselbe Philosophie verfolgt wie Backbone, spricht es dieselbe Zielgruppe an: Wer primär in zwischen Webclient und -server synchronisierten Entitäten und Listen denkt, verfügt mit Ampersand über eine moderne Alternative zu Backbone. Der große Vorteil gegenüber anderen Frameworks ist, dass Entwickler auf möglicherweise aus Backbone bereits vorhandenes Wissen aufbauen können, sodass sie sich nicht komplett neu einarbeiten müssen. Eine sanfte Migration von Backbone zu

Ampersand ist ebenfalls denkbar und mit geringem Aufwand machbar.

Ember.js

Zumindest konzeptionell unterscheidet sich Ember.js von den bislang vorgestellten Frameworks deutlich, da es einen vollständigen, integrierten Ansatz zur Entwicklung clientseitiger Anwendungen darstellt. Es unterstützt bidirektionale Datenbindung und das Verwalten von Ansichten ebenso wie das Modellieren von Entitäten, das Synchronisieren zwischen Webclient und -server sowie clientseitiges Routing.

Vorkenntnisse in Backbone erleichtern den Einstieg auch in Ember, da sich viele grundlegende Konzepte ähneln. Vor allem der Umgang mit Entitäten, die Ember als „Model“ bezeichnet, erinnert deutlich an Backbone. Das gilt auch für die Persistenz, für die ähnliche Funktionen wie in Backbone zur Verfügung stehen.

Ember kennt im Gegensatz zu Backbone das Konzept des Controllers, allerdings ähneln Controller hier den Views von Backbone enorm. Der Hauptunter-

schied besteht in der Verfügbarkeit der Datenbindung, weshalb Entwickler mit Ember kein zusätzliches Modul eines Drittanbieters zur Verwaltung der Ansichten benötigen. Als Sprache für die Views setzt das Framework gänzlich auf das ebenfalls eigenständig verfügbare Modul Handlebars.

Besonderes Augenmerk verdient das Modul Ember Data, das die Synchronisation von Daten zwischen Webclient und -server übernimmt. Es ähnelt einem O/R-Mapper (objektrelationaler Mapper) und kann Relationen zwischen Entitäten verwalten, was die Arbeit mit Daten im Vergleich zu Backbone deutlich vereinfacht. Außerdem ist Ember Data nicht auf REST-basierte Webdienste beschränkt. Adapter ermöglichen prinzipiell das Anpassen des Moduls an beliebige Persistenztechniken und Protokolle, beispielsweise Websockets.

Für Chrome und Firefox steht zudem ein Add-on namens Ember-Inspector zur Verfügung, das das Debugging von Ember-Anwendungen vereinfacht.

Ember gelingt es, die Brücke zwischen Backbone und der Moderne zu schlagen. Einen Großteil der Komplexität

von Backbone vermeidet dieses Framework, indem es häufig auf das Konzept „Convention over Configuration“ setzt. Das verringert die Codegröße, erschwert den Einstieg für Neulinge aber erheblich, da häufig nicht klar ist, wie einzelne Komponenten zusammenhängen oder wie sie interagieren.

AngularJS

Das von Google entwickelte AngularJS ist wie Ember eine integrierte Entwicklungs-Umgebung, die alles enthält, was zur Programmierung von Single-Page-Anwendungen erforderlich ist. Sein Schwerpunkt liegt aber deutlich weniger auf JavaScript-Code, sondern vielmehr auf den deklarativen Fähigkeiten von HTML. Angular würde unter der Fragestellung entwickelt, wie HTML aussehen würde, wenn man es von vornherein für die Entwicklung von Single-Page-Anwendungen entworfen hätte.

Angular greift diese Idee auf, indem es das Erstellen benutzerdefinierter HTML-Elemente und -Attribute ermöglicht. Auf diesem Weg kann der Entwickler HTML nach Bedarf erweitern, was vor allem die

Server- und Storage systeme kauf man am besten beim Profi.

www.rnt.de

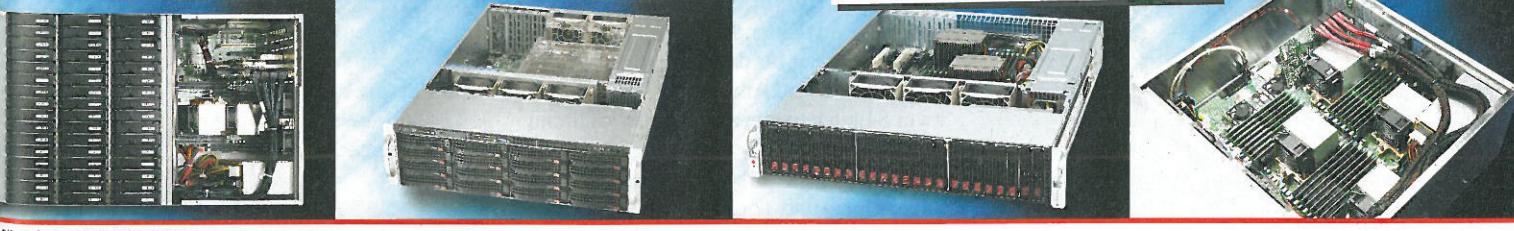
Egal, ob als **Datenbankserver**, **Enterprise Storage server**, **Nearline Storage** oder als **Virtual Tape Library zur Langzeitarchivierung**, mit Server- und Storage systemen von Rausch Netzwerktechnik bekommen Sie viel zu einem kleinen Preis. Durch die flexiblen Möglichkeiten sind vielfältige Anwendungen möglich. Wir bieten verschiedene Basiskonfigurationen an, die Sie an Ihre jeweiligen Anforderungen anpassen können. Gerne beraten wir Sie.

Beispiele:
2HE - 24x 2,5", max. 28,8 TB
3HE - 16x 3,5", max. 96 TB
4HE - 48x 3,5", max. 288 TB

Weitere Informationen erhalten Sie im Internet unter www.rnt.de oder gerne telefonisch unter 0800 5929-100*



1. Datacenter ICT Storage Hardware Product of the Year
2. Datacenter ICT Server Product of the Year



*Kostenlos aus dem deutschen Festnetz.



Rausch Netzwerktechnik GmbH

Englerstraße 26 · D-72675 Ettlingen
 Telefon (07243) 5929-0 · Telefax -14 · info@rnt.de
www.rnt.de



RAUSCH NETZWERKTECHNIK

www.rnt.de

Sympathisch und gut beraten. Bestens betreut.

Zusammenarbeit mit Designern erleichtert, da die Semantik und nicht die technische Lösung in den Vordergrund rückt. Die Zeile

```
<calendar view="year" current-year="2014" />
    show-week-numbers="true" />
```

veranschaulicht das. Da Angular die bidirektionale Datenbindung zwischen HTML und JavaScript auf elegantem Wege unterstützt, lassen sich zudem leicht Controller entwickeln, die auf Änderungen in der Ansicht reagieren. Listing 3 zeigt ein Beispiel.

Logik, die verschiedene Controller gemeinsam verwenden, lagert Angular in sogenannte Services aus, die sich innerhalb einer Anwendung als Singleton-Objekte verhalten. Serienmäßig enthält das Framework eine Reihe solcher Dienste, beispielsweise zur einfachen Kommunikation über HTTP.

Das Framework sieht zur Organisation von Code ein eigenes Modulsystem mit einer integrierten Abhängigkeitsverwaltung vor. Für die Entkopplung von Modulen und Komponenten sowie der Kommunikation zwischen ihnen stellt Angular einen leistungsstarken Event-Mechanismus zur Verfügung, der die Basis auch für umfangreiche und komplexe Anwendungen bildet.

Von allen hier vorgestellten Frameworks wirkt AngularJS mit Abstand am modernsten, da Google bereit war, Bestehendes zu hinterfragen und HTML von Grund auf zu überdenken. Das zahlt sich bei der Entwicklung aus, da Angular die Leichtigkeit von HTML erhält, sie aber mit der Leistung eines vollständigen MVC-Frameworks kombiniert. So gelingt der Spagat, für Einsteiger leicht verständlich zu sein, ohne komplexe, professionelle Anwendungsentwicklung zu behindern.

Letztlich nimmt das Framework viele zukünftige Konzepte von HTML vorweg. Der Fokus liegt auf dem Erzeugen eigener HTML-Elemente und -Attribute, und die deklarative Entwicklung sowie die Entkopplung von Komponenten entsprechen ungefähr der Idee der Web Compo-

Onlinequellen

ECMAScript 6, voraussichtlicher Termin	twitter.com/awbjs/status/474662357516689410
Data-binding Revolutions with <i>Object.observe()</i>	www.html5rocks.com/en/tutorials/es7/observe/
Using React components as Backbone Views	www.thomasboyt.com/2013/12/17/using-reactjs-as-a-backbone-view.html
Faster AngularJS Rendering (AngularJS and ReactJS)	www.williambrownstreet.net/blog/2014/04/faster-angularjs-rendering-angularjs-and-reactjs/
<i>npm</i> -Module für Ampersand.js	tools.ampersandjs.com/
Ember Data	github.com/emberjs/data
Ember-Inspector	github.com/emberjs/ember-inspector
Handlebars Templates	handlebarsjs.com/
Introduction to Web Components	www.w3.org/TR/2013/WD-components-intro-20130606/
Dimitri Glazkov; What the Heck is Shadow DOM?	glazkov.com/2011/01/14/what-the-heck-is-shadow-dom/
ngReact.js	davidchang.github.io/ngReact/

nents und des Shadow DOM – wenn auch mit einigen Abstrichen.

Im direkten Vergleich zu den übrigen Frameworks kann Angular in nahezu jeglicher Hinsicht glänzen. Letztlich verheiratet es die besten Ideen aus allen Welten und stattet sie mit einem äußerst leistungsfähigen, skalierbaren und zeitgemäßen Unterbau aus. Das Framework eignet sich daher für kleine und große Webanwendungen gleichermaßen, weshalb es kaum einen Grund gibt, auf seinen Einsatz zu verzichten.

Wie erwähnt, ist auch React für große und komplexe Webanwendungen gegebenfalls eine sinnvolle Ergänzung in Bezug auf die Performance der Darstellung. Eine Integration von React in Angular ist bereits verfügbar.

Fazit

Die sechs JavaScript-Frameworks Knockout, React, Backbone, Ampersand, Ember und Angular haben zwar ein gemeinsames Ziel, unterscheiden sich aber in ihrer Philosophie und ihren Konzepten grundlegend voneinander. Während die Hochzeit von Knockout vorüber ist, eignet es sich nach wie vor für kleine Webanwendungen, die nur eine einzige Ansicht enthalten. Allerdings lassen sich derartige Anwendungen ähnlich einfach mit Angular entwickeln. Bei dessen Verwendung erhält der Entwickler aber ein deutlich höheres Potenzial für den späteren Ausbau.

Backbone und Ampersand sind letztlich zwei Seiten derselben Medaille: Während Backbone übermäßig komplex und nicht mehr zeitgemäß wirkt, ist Ampersand eine angenehme und überfällige Modernisierung von Backbone. Allerdings wird sich

zeigen müssen, ob Ampersand langfristig gegen die Konkurrenz bestehen kann. Das steht und fällt nicht zuletzt mit der umfangreichen Unterstützung der Community, die zumindest derzeit noch aussteht.

Ember und Angular sind die einzigen Komplettsolutions, die auch Dependency Injection unterstützen, also die Abhängigkeiten eines Objekts zur Laufzeit verfolgen. In ihren Konzepten unterscheiden sie sich jedoch grundlegend. Ember zeigt deutliche Anleihen an Backbone und ist durch sein „Convention over Configuration“-Konzept für Einsteiger oft verwirrend.

Angular hingegen ist gleichermaßen verständlich und leistungsfähig. Es beweist, dass es sich gelegentlich durchaus lohnt, Bestehendes zu hinterfragen und den Mut zu haben, gänzlich neue Wege einzuschlagen. Wer heute auf der Suche nach einem MVC-Framework zur Entwicklung clientseitiger Anwendungen ist, kommt um Angular de facto nicht herum.

React tanzt zu guter Letzt aus der Reihe, da es sich hierbei um kein MVC-Framework handelt, sondern lediglich um einen Weg, Ansichten performant darzustellen. Da dies aber bei komplexen Ansichten ein Schwachpunkt aller vorgestellten Frameworks ist, bietet sich React als Ergänzung an, unabhängig davon, für welches MVC-Framework man sich entscheidet. (ka)

Golo Roden

ist Gründer der „the native web UG“, eines auf native Webanwendungen spezialisierten Unternehmens.

