JAVA Mag

Java magazin

Java | Architektur | Software-Innovation

Das JavaScript-Ökosystem

Die wichtigsten Tools, Trends und Frameworks

JS

Webentwicklung mit JavaScript: Vergangenheit, Gegenwart und Zukunft

Eine kleine Zeitreise durch das JavaScript-Ökosystem

Bei keiner anderen Programmiersprache gab es in den vergangenen Jahren so viele Hypes, Trends und neue Frameworks. Im JavaScript-Bereich die Übersicht zu behalten, fällt durch diese Schnelllebigkeit schwer. Ein Blick in die Historie kann hier hilfreich sein, denn es wird deutlich, welche Entwicklung das JS-Ökosystem seit seinen Anfängen vor mehr als zwanzig Jahren genommen hat und wie aktuelle Technologien „evolutionär“ entstanden sind.

von Philip Stroh

Der vorliegende Artikel ist der Versuch, das JavaScript-Ökosystem nicht allein auf Basis der brandneuesten Technologien darzustellen, sondern dessen Entwicklung von seinen Anfängen bis zu den heute gängigen Frameworks einmal aus der Historie zu betrachten. Der Artikel ist im Rahmen einer JavaScript-Frameworkevaluierung entstanden. Dabei lag der Schwerpunkt hauptsächlich auf mit JavaScript entwickelten Webanwendungen. Themen wie serverseitiges JavaScript und Mobile Apps werden aus diesem Grund nur am Rande betrachtet. Aufgrund der unzähligen JavaScript-Frameworks und -Bibliotheken erhebt dieser Artikel keinen Anspruch auf Vollständigkeit – es geht vielmehr darum, Trends, Impulse und Fortschritte im Bereich der Webentwicklung anhand wichtiger oder signifikanter Vertreter darzustellen. Einzelne Themen und Frameworks werden meist nur kurz angerissen, für detailliertere Informationen ist unter [1] eine umfangreiche Quellensammlung hinterlegt.

Abbildung 1 zeigt die ausgewählten JavaScript-Frameworks, -Features, -Tools und -Trends auf einem Zeitstrahl. Die zugrunde gelegten initialen Releasedaten wurden größtenteils Wikipedia entnommen.

Es war einmal ...

Die Geschichte von JavaScript beginnt im Mai 1995. Der Browserkrieg zwischen Netscape und Microsoft ist gerade in vollem Gang. Um Marktanteile zu gewinnen bzw. zu behaupten, geht es darum, die Browser möglichst schnell mit weiteren Features auszustatten. Brendan Eich, später einer der Gründer des Mozilla-Projekts, entwirft für Netscape in nur zehn Tagen die erste prototypische Version einer Skriptsprache zur Ausführung im Browser. Die Sprache besteht aus Anleihen aus Scheme (funktionale Programmierung), Self (OO-Programmierung) und Java (Syntax). Zuerst Mocha und LiveScript genannt, wird die Sprache im Dezember 1995 – wohl primär aus Marketinggründen – in JavaScript umgetauft. Im März 1996 erscheint der Browser Netscape 2.0 mit JavaScript-Support. Es gab damit erstmals eine Skriptsprache, um dynamisch und clientseitig auf HTML-Inhalte zuzugreifen.

Im gleichen Jahr veröffentlicht Microsoft mit JScript eine Entsprechung für den Internet Explorer (IE). Um die Sprachvarianten zu vereinheitlichen, wird JavaScript in der Folge unter dem Namen ECMAScript standardisiert. JavaScript, JScript und auch ActionScript von Adobe implementieren die ECMAScript-Spezifikation.

1999 wird die ECMAScript-Spezifikation in der 3rd Edition veröffentlicht. Diese Version wird auch als „Baseline“ für das heutige JavaScript bezeichnet. Folgende Bemühungen, JavaScript weiter zu entwickeln, scheitern in den nächsten Jahren vorerst. Erst zehn Jahre später gibt es mit ES5 eine neue Version des Standards.

Ajax und jQuery

2005 wird Ajax (Asynchronous JavaScript and XML) zum Hypethema. Ajax beschreibt einen Ansatz, von einer Webseite asynchron Daten vom Server anzufragen und anzuzeigen, ohne die Seite komplett neu zu laden. Der DOM-Baum der HTML-Seite wird dabei per JavaScript entsprechend der vom Server geladenen Daten partiell verändert. Es entstehen daher entsprechende Bibliotheken, um DOM-Manipulationen per JavaScript einfach durchführen zu können, darunter Dojo, YUI und jQuery. Erste Anwendungen mit großer Nutzerzahl wie Yahoo Mail, Google Maps und Google Mail setzen das Konzept um.

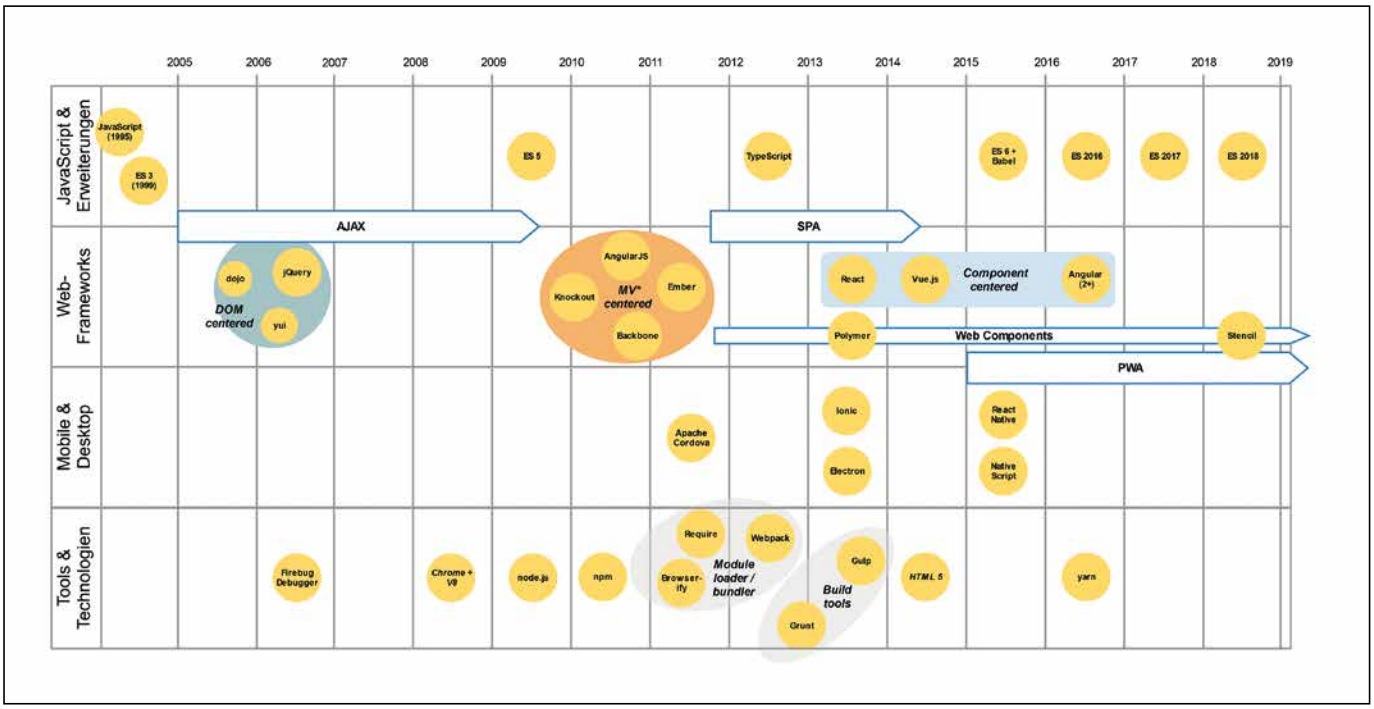

Abb. 1: Zeittafel signifikanter JavaScript-Frameworks, -Features, -Tools und -Trends

JQuery wird in den folgenden Jahren zur meistverwendeten JavaScript-Bibliothek und ist es bis heute. Laut [2] setzen aktuell 80 % aller Top-10 000-Webseiten jQuery ein. Hauptfeatures von jQuery sind die einfache DOM-Selektion, -Navigation und -Manipulation per JavaScript, die es ermöglichen, einfach Ajax-Webanwendungen zu bauen. Für die DOM-Zugriffe müssen zu der Zeit die Eigenheiten der verschiedene Browser, u. a. des IE 6, berücksichtigt werden. jQuery löst diese Problematik durch ein einheitliches API. Ajax und jQuery verändern die Webentwicklung grundlegend. Die Konzepte bilden die Grundlage für heutige Single-page Applications (SPA). In der Folge wird JavaScript zur Entwicklung von Webseiten und -anwendungen deutlich populärer. Um die Entwicklungsmöglichkeiten zu verbessern, entstand u. a. Tooling wie der Firebug Debugger, der es ermöglicht, JavaScript-Code im Browser zur Laufzeit zu analysieren.

JavaScript auf dem Server

2008 steigt Google mit Chrome in den Browsermarkt ein. Zu der Zeit ist IE 8 der mit Abstand meistgenutzte Browser. Heute, zehn Jahre später, hat sich das Bild gedreht und Chrome ist zum Browser mit den größten Marktanteilen aufgestiegen. Die Kernelemente von Chrome, u. a. die Rendering-Engine Blink und die JavaScript Engine V8, werden von Google als Open-Source-Projekt unter dem Namen Chromium bereitgestellt.

2009 entwickelt Ryan Dahl auf Basis von V8 Node.js eine Laufzeitumgebung zur serverseitigen Ausführung von JavaScript. Mit Node.js stellt er die Multi-Thread-Architektur der gängigen Webserver infrage. Statt den Request-Thread bspw. bei I/O-Zugriffen zu blockieren, setzt Node.js auf eine nichtblockierende, eventgesteuerte Architektur, bei der ein einziger Thread für die Verarbeitung der eingehenden Requests zuständig ist. Die Idee, serverseitig mit JavaScript zu entwickeln, war nicht neu. Bereits Netscape veröffentlichte 1997 mit Rhino eine in Java geschriebene JavaScript Engine. Allerdings wurde mit dem beschriebenen Aufbau von Node und der Maschinencodekompilierung von V8 eine sehr viel bessere Performance erreicht.

Weitere Erfolgsfaktoren waren die stark wachsende JavaScript-Community und ab 2010 npm, der Package-Manager für Node.js. Per npm wird es sehr einfach, JavaScript-Bibliotheken bereitzustellen und einzubinden. Heute enthält das npm-Repository fast 800 000 Packages. Wöchentlich werden mehr als fünf Millarden Downloads registriert. Auf Basis von Node.js lassen sich z. B. mit Frameworks wie Express Serverapplikationen mit JavaScript entwickeln. Node.js und npm sind aber auch die Basis für die weitere Entwicklung der clientseitigen Frameworks. Diese werden nach 2010 als npm-Pakete bereitgestellt, und Node.js dient als Laufzeitumgebung für Build- und Bundle-Prozesse oder zur Ausführung eines lokalen HTTP-Servers während der Entwicklung.

ES5 (ECMAScript 5th Edition) mit Verbesserungen der Sicherheit und Stabilität wird ebenfalls 2010 veröffentlicht. ES5 führt u. a. den Strict Mode und JSON-Support ein. Ein wesentlich ambitionierterer Versuch, Klassen, ein Modulsystem und Typisierung mit einer vierten Version des Standards einzuführen, war zuvor aufgegeben worden. ES5 ist heute in allen gängigen Browsern implementiert.

MV*-Frameworks

Mit der Zeit werden Applikationen auf Basis von jQuery oder ähnlichen Frameworks immer größer. Dabei fehlen aber Patterns zur Strukturierung der Anwendungen. Um 2010 kommen verschiedene MV*-Frameworks auf, um größere Ajax-Anwendungen zu strukturieren und wartbarer zu gestalten – darunter Backbone, Knockout, AngularJS und Ember. Kernfunktionen dieser Frameworks sind ein Data Binding zwischen Datenobjekten (View Model) und View sowie ein HTML Templating. Das Data Binding sorgt dafür, dass Änderungen zwischen Datenobjekten und View automatisch aktualisiert werden. Die meisten Frameworks stellen auch ein Routing zur Verfügung, über das sich Browser-URL und -History steuern lassen und somit Back-Button und Bookmarks durch die Anwendungen unterstützt werden.

Für Applikationen auf Basis dieser Frameworks wird der Begriff Single-Page Applications geläufig. Googles AngularJS wird der populärste Vertreter zur Entwicklung von SPAs. Die Anwendungen werden dabei durch Controller, Direktiven und Services strukturiert. Direktiven erlauben es bspw., bestimmte HTML-Fragmente als wiederverwendbare Komponenten zu kapseln.

Um Anwendungen und Bibliotheken wartbar halten zu können, und zur Vermeidung von Namenskonflikten, entstehen die Modulsysteme bzw. -stile CommonJS (synchron, Node.js) und AMD (asynchron) sowie entsprechende Module Loader und Bundler, z. B. Browserify (CommonJS), Require.js (AMD) und webpack. Über die Task Runners Grunt und Gulp wird der Build-Prozess für die komplexer werdenden Applikationen konfiguriert.

2012 veröffentlicht Microsoft eine erste Version von TypeScript. TypeScript ist ein Superset von JavaScript, d. h. jedes gültige JavaScript-Programm ist auch ein syntaktisch gültiges TypeScript-Programm. TypeScript führt ein optionales Typensystem und Spracherweiterungen wie Module, Klassen und Interfaces ein und kann per Compiler nach JavaScript transpiliert werden.

JavaScript überall

Neben der reinen Webentwicklung etabliert sich JavaScript zu der Zeit auch im Bereich Mobile- und Desktopentwicklung.

Adobe stellt 2011 Apache Cordova als Open Source bereit. Mit Cordova lassen sich Cross-Plattform-Apps für Mobilgeräte mit den Webtechnologien JavaScript, HTML5 und CSS entwickeln. Cordova wrappt diese Apps für die jeweilige Plattform und bietet per Plug-ins

Zugriff auf native Funktionen. Für das Rendering wird die WebView des Mobilgeräts genutzt. Diese hybriden Apps – mit Web- und nativen Anteilen – können über den jeweiligen App Store bereitgestellt werden. Ein populäres Framework, das auf Cordova aufsetzt, ist Ionic. Es wird 2013 veröffentlich und ermöglicht die Entwicklung von hybriden Apps auf Basis von AngularJS bzw. Angular.

Auch für den Desktop lassen sich mit JavaScript Anwendungen entwickeln. Hierzu veröffentlicht GitHub 2013 Electron. Zuerst unter dem Namen Atom Shell herausgebracht, wird es von GitHub eingesetzt, um den Codeeditor Atom zu bauen. Electron kombiniert Chromium für das Rendering der Webseiten mit Node.js zum Zugriff auf Funktionen des Betriebsystems. Durch diese Architektur können mit Electron Cross-Plattform-Apps für Windows, Mac und Linux entwickelt werden. Gleichzeitig führt dieser Aufbau allerdings auch – im Vergleich zu nativen Anwendungen – zu größeren Anwendungs-Bundles und einem höheren Ressourcenverbrauch. Auf Basis von Electron baut Microsoft ferner den Editor Visual Studio Code. Auch die Desktopvarianten von Messengern wie Skype, WhatsApp, Slack oder Mattermost sind mit Electron entwickelt [3].

Komponentenzentrierte Frameworks

2013 veröffentlicht Facebook React als Open-Source-Projekt, nachdem bereits seit 2011 erste Teile von Facebook damit entwickelt wurden. Im Unterschied zu bisherigen MV*-Frameworks sind bei React Komponenten der zentrale Architekturbestandteil. Komponenten kapseln Logik und Darstellung eines Widgets oder eines bestimmten Bereichs einer Webseite. Die Oberflächen einer React-Anwendung werden aus Komponentenhierarchien erstellt. React führt verschiedene neue Konzepte ein, darunter Virtual DOM zum performanten Updaten des DOMs und JSX, eine JavaScript-Erweiterung, die es erlaubt, HTML-Syntax direkt in JavaScript-Funktionen zu verwenden.

Auch die weiteren neuen Frameworks Angular und Vue.js sind wie React komponentenzentriert. Angular (lange auch Angular 2 genannt) ist eine nicht abwärtskompatible Neuentwicklung vom AngularJS-Team (Google). Es ist in TypeScript geschrieben und empfiehlt TypeScript zur Entwicklung von Anwendungen mit dem Framework. Während die React-Bibliothek sich allein auf den View-Layer fokussiert, bietet das Angular-Framework out of the Box einen breiten Funktionsumfang, z. B. Routing, Dependency Injection und einen HTTP-Client. Es setzt u. a. auf HTML-Templates und ein 2-Way Data Binding. Vue.js wurde vom ehemaligen Google-Entwickler Evan You implementiert und wird von ihm als Communityprojekt geführt. Vue kombiniert Ansätze von React (z. B. Virtual DOM und unidirektionalen Datenfluss) und Angular (HTML-Templates).

Mit React Native und Native Script (Angular, Vue) entstehen Ansätze, mobile Apps zu entwickeln und dabei ohne die WebView, d. h. ohne HTML/Rendering auszukommen. Stattdessen werden Komponenten direkt als native UI-Elemente gerendert.

Nach Jahren der Stagnation wird 2015 die ES-Version 6, auch ES 2015 genannt, veröffentlicht. Der ECMAScript-Standard soll nun jährliche Updates erfahren (daher werden in der Folge ES 2016, 2017 und 2018 veröffentlicht). In ES6 werden Module, Klassen und viele weitere Sprachfeatures (wie Promises, Arrow Functions, Destructuring, Block-scoped Variables, Collections ...) eingeführt. ES 2017 führt zum einfacheren Umgang mit asynchronen Aufrufen async/await ein. Mit dem Compiler Babel ist es möglich, neuere ES-Versionen – ähnlich wie beim TypeScript-Compiler – in ES5 zu übersetzen. Somit kann mit den neuen Sprachfeatures entwickelt werden, obwohl sie nicht in allen Browsern unterstützt werden. „Problembrowser“ ist hier insbesondere der IE 11, der bis heute fast keins der ES-6-Sprachfeatures unterstützt. Auch TypeScript implementiert alle neuen standardisierten ES-Features. Neue ECMAScript-Standards werden durch Technical Commitee 39 (TC39) spezifiziert. Der Spezifikationsprozess sieht fünf Stufen (Stages 0–4) vor. Neu vorgeschlagene Sprachfeatures und deren aktuelle Einstufung können unter [4] eingesehen werden.

Webentwicklung mit JavaScript heute

Nach einer Phase ständiger Neuerungen scheinen sich aktuell in einigen Bereichen stabilere Trends auszubilden. Stand heute können folgende Frameworks und Tools als De-facto-Standards bei der Entwicklung von Webanwendungen angesehen werden:

- *JavaScript:* Bei neuen Projekten kommt quasi durchgehend JavaScript in der aktuellsten ES-Version mit Babel oder TypeScript zum Einsatz. TypeScript hat sich u. a. durch Angular in der JavaScript-Community stark verbreitet, auch React und Vue.js bieten TypeScript-Support. Allerdings sind sie im Gegensatz zu Angular nicht in TypeScript entwickelt. Eine Alternative zu TypeScript mit weniger Verbreitung ist Facebooks Flow.
- *Tooling (Build, Bundler, Package-Manager):* Bundler wie RequireJS oder Browserify spielen fast keine Rolle mehr. Stattdessen kommt in Verbindung mit allen neuen Webframeworks und dem ES-Modulsystem hauptsächlich webpack als Bundler zum Einsatz. Meist wird webpack dabei nicht direkt genutzt, sondern das jeweilige Command Line Interface (CLI). Da die webpack-Konfiguration relativ kompliziert und umfangreich werden kann, kapseln diese Tools die webpack-Funktionalität, um die Nutzung zu vereinfachen. Über das CLI von Vue, React und Angular lassen sich schnell vorkonfigurierte Projekte aufsetzen, sodass direkt mit der Entwicklung gestartet werden kann. Statt der Build-Tools Grunt und Gulp wird mittlerweile meist die npm-Funktion *npm scripts* als Task-Runner für den Build- und Entwicklungsprozess

verwendet. Als Package-Manager kommt hauptsächlich npm zum Einsatz. Die von Facebook entwickelte, kompatible npm-Alternative Yarn wird aufgrund von Skalierungsproblemen mit npm in großen Projekten immer häufiger eingesetzt.

- *Webframeworks/-bibliotheken:* Die alten MV*-Frameworks Knockout, Backbone und AngularJS spielen für neue Projekte keine Rolle mehr. Frühere Backbone-User wie Airbnb und Pinterest wechseln zu React [5], [6]. AngularJS ist durch Angular abgelöst, erhält aber noch für die nächsten drei Jahre Long-Term-Support. Gehalten hat sich von dieser Frameworkgeneration nur noch Ember. LinkedIn bspw. ist mit Ember entwickelt. In der Verbreitung hat es sich aber insgesamt gegenüber den aktuellen Frameworks nicht durchgesetzt.

Neue Projekte werden heute hauptsächlich mit den moderneren, komponentenzentrierten Webbibliotheken React, Angular oder Vue entwickelt. Aktuell ist React hier am stärksten verbreitet. Neben Facebook und Instagram wird es u. a. auch von Netflix, Twitter Mobile, Uber, Atlassian, Slack und vielen weiteren großen Webfirmen eingesetzt [7]. Auch Microsoft entwickelt bspw. www.outlook.com mit React und TypeScript. Facebook selbst hat über 50 000 Komponenten mit React implementiert [8]. Angular wird von Google mittlerweile in erheblichem Umfang genutzt, aktuell in 600 Anwendungen [9]. Bei großen Internetunternehmen gibt es sonst wenige Referenzen für den Einsatz von Angular [10]. Angular scheint eher bei der Umsetzung von SPAs im Enterprise-Kontext beliebt. Zwei bekannte Beispiele für AngularJS-Anwendungen – Kibana und Grafana – planen keinen Wechsel auf das neue Angular, sie steigen stattdessen auf React um [11], [12]. Das dritte Komponentenframework Vue hat eine geringere Verbreitung, gewinnt insgesamt aber an Popularität. Ein bekanntes Projekt, das Vue einsetzt, ist bspw. GitLab [13].

Alle drei Frameworks versprechen eine stabile Weiterentwicklung ihrer APIs. Referenzen zur Versionsstrategie von Angular und React sind unter [14] und [15] hinterlegt. Bei Angular und React stehen in den kommenden Major-Versionen spannende neue Features an. Bei Angular ist das u. a. der Ivy-Renderer, der die Bundle-Size von Angular stark reduzieren soll, und bei React das Async Rendering (Time Slice und Suspense).

Web Components und PWAs

Zwei weitere Themen stellen wir an das Ende des Artikels. Nicht wirklich neu, haben sie heute das Potenzial, die Frontend-Entwicklung mit JavaScript in den nächsten Jahren stark zu verändern: Web Components und Progressive Web Apps.

Web Components sind kein neues Thema, die Idee wurde bereits 2011 initial vorgestellt. In der Folge wurden die Spezifikation und das Framework Polymer entwickelt. Die Idee hinter Web Components ist es, UI-Widgets wie bspw. einen Date Picker ohne Frameworkabhängigkeiten mit Standard-Webtechnologien bereitstellen zu können. Hierzu wurden unter dem Oberbegriff Web Components mehrere Technologien für HTML und DOM vom W3C spezifiziert: Custom Elements (= Definition eigener HTML-Tags), Shadow DOM (= Isolierung von DOM und CSS, um Namenskonflikte zu vermeiden), HTML Imports (= Import von HTML-Dokumenten in HTML-Dokumente) und HTML-Templates. Diese Standards sind aktuell nicht in allen Browsern implementiert, es existieren jedoch Polyfills für alle gängigen Browserversionen. Google ist mit Chrome und dem Framework Polymer einer der Treiber dieser Idee. Polymer ist ein Webframework zur Entwicklung von Anwendungen auf Basis von Web Components. Es wird von Google genutzt, hat sich aber bisher nicht in der Breite durchgesetzt.

Seit 2017 erhält das Thema neuen Schwung, da mit Stencil und Angular Elements zwei Bibliotheken zur Erstellung von Web Components angekündigt wurden. Stencil ist ein Framework der Ionic-Macher. Es wurde u. a. entwickelt, um mit Ionic auch Progessive Web Apps umsetzen zu können – hierzu gleich mehr. Ionic Version 4 – eine erste Betaversion ist seit Juli verfügbar – wird unabhängig von Angular sein, d. h. Ionic-Apps sollen auch mit React oder Vue.js entwickelt werden können. Alle Ionic-Komponenten werden hierzu mit Stencil als Web Components bereitgestellt. Stencil setzt dabei in der Entwicklung auf Ansätze von React (Virtual DOM und JSX) in Kombination mit TypeScript. Die Komponenten werden durch den Stencil-Compiler in VanillaJS (d. h. reines JavaScript ohne Library-Abhängigkeiten) übersetzt und somit möglichst klein gehalten.

Angular Elements soll es ermöglichen, Angular-Anwendungen als Web Components zu erstellen. Ein Baustein hierzu ist der erwähnte Ivy Renderer, der zur Reduktion der Bundlegröße der Komponente notwendig ist. Nach aktueller Einschätzung werden Web Components durch Stencil und evtl. auch Angular Elements praktikabler und werden zukünftig weitere Verbreitung finden. Insbesondere für die Hersteller von UI-Komponenten neben Ionic bspw. PrimeTek oder Kendo UI ist dieser Ansatz sehr interessant, da diese ihre Komponenten bisher speziell für die verschiedenen Frameworks entwickeln müssen. Ein weiterer Anwendungsfall für Web Components sind Micro Frontends, d. h. Frontends, die sich ähnlich dem Microservices-Ansatz im Backend aus verschiedenen kleineren Views oder Widgets zusammensetzen.

Der Begriff Progressive Web Apps kam 2015 auf. Er beschreibt eine neue Art, Apps insbesondere für mobile Geräte zu entwickeln, die ohne einen App Store über den Browser installiert werden können. Mit hybriden Apps gibt es, wie beschrieben, bereits die Möglichkeit, Cross-Plattform-Apps mit den Webtechnologien HTML, JavaScript und CSS zu entwickeln und in den jeweiligen App Stores bereitzustellen. Auch PWAs sind keine nativen, sondern Web-Apps. Allerdings sind sie nicht per Cordova etc. gewrappt und müssen

auch nicht über einen App Store installiert werden. Stattdessen laufen diese Apps direkt im Browser des Mobilgeräts und werden über den Browser installiert. Installieren heißt in diesem Fall, dass auf dem Homescreen des Geräts ein Icon platziert wird, über das die Web-App erneut aufgerufen werden kann. Über die von W3C standardisierten Service Worker sind PWAs im Browser offlinefähig und können Push Notifications auf den Geräten auslösen, auch wenn der Browser durch den User beendet wurde.

Google treibt seit 2015 die Entwicklung von PWAs an. PWAs werden daher bereits seit einiger Zeit von Android und Chrome unterstützt. Im Frühjahr 2018 wurde auch eine Safari-Version für iOS mit Support für Service Worker und das Web App Manifest veröffentlicht – hier gibt es allerdings noch einige Einschränkungen im Vergleich zur Nutzung unter Android. Neben Google setzt auch Microsoft auf PWAs und unterstützt seit dem April-2018-Update von Windows 10 und Edge auch Push Notifications. Das bedeutet, absehbar sind PWAs auf allen Plattformen (nicht nur auf mobilen Geräten) möglich. Die Ionic-Macher veröffentlichten Anfang 2018 die Betaversion eines PWA-Toolkits mit Ionic, Stencil, Offlinesupport und Web Push Notifications. PWAs etablieren sich somit als weitere Möglichkeit, Cross-Plattform-Apps für mobile Geräte und den Desktop mit JavaScript zu entwickeln.

Fazit

Insbesondere in den letzten zehn Jahren hat sich die Frontend-Entwicklung mit JavaScript sehr schnell und evolutionär weiterentwickelt. Einige der kurzzeitig gehypten Tools sind dabei schon wieder passé und werden nicht mehr für neue Projekte eingesetzt. Für langfristig angelegte Projekte ist es in diesem Umfeld schwierig, die Zukunftssicherheit einer gewählten Technologie abzuschätzen. In jedem Fall sollten für die Technologiewahl die Verbreitung des Frameworks und der Umfang des Einsatzes als ausschlaggebende Kriterien berücksichtigt werden. Gleichzeitig ist bei der Umsetzung von Applikationen sorgfältig abzuwägen, an welchen Stellen und zu welchem Grad man sich von einem Framework abhängig machen will.

Zurzeit scheint sich das Ökosystem, u. a. auch durch die Dominanz von Angular und React, etwas zu stabilisieren. Beide Technologien werden dabei mit großem Augenmerk auf Abwärtskompatibilität kontinuierlich weiterentwickelt. Es zeichnen sich aber auch neue Trends ab, die zu neuen Möglichkeiten in der Frontend-Entwicklung führen werden. Nach dem DOM-, MV*- und komponentenzentrierten Frameworks steht durch die Fortschritte im Bereich Web Components und PWAs die nächste Evolutionsstufe an.

Philip Stroh arbeitet als Softwarearchitekt bei der TIMOCOM GmbH. Er hat langjährige Erfahrung in der Entwicklung und Konzeption von Webanwendungen mit Java und JavaScript. Seine weiteren Schwerpunkte sind NoSQL-Datenbanken und Delivery-Automatisierung.

Links & Literatur

[1] https://github.com/pstrh/pub/

[2] https://trends.builtwith.com/javascript/jQuery

[3] https://electronjs.org/apps

[4] https://github.com/tc39/proposals/

[5] https://de.slideshare.net/spikebrehm/the-evolution-of-airbnbs-frontend

[6] https://medium.com/@Pinterest_Engineering/migrating-pinterest-profiles-to-react-479f4f7306aa

[7] https://github.com/facebook/react/wiki/sites-using-react/

[8] https://reactjs.org/blog/2018/03/27/update-on-async-rendering.html

[9] https://medium.com/@jkeung/google-i-o-2018-whats-new-in-angular-611cc21befcf

[10] https://www.madewithangular.com

[11] https://github.com/elastic/kibana/issues/10271

[12] https://github.com/grafana/grafana/wiki/Frontend:-Code-structure,-architecture-and-plans-for-the-future

[13] https://github.com/vuejs/awesome-vue/

[14] https://angular.io/guide/releases/

[15] https://reactjs.org/blog/2016/02/19/new-versioning-scheme.html

Internationalisierung von Single Page Applications

Around the World

Für Webapplikationen, die von Anwendern aus verschiedenen Ländern genutzt werden, ist Internationalisierung eine Standardanforderung. Wir wollen verschiedene Ansätze betrachten, die es ermöglichen, eine JavaScript-Applikation an die jeweiligen sprach- und landesspezifischen Besonderheiten anzupassen. Exemplarisch wird eine React-Anwendung internationalisiert.

von Philip Stroh

Zu Beginn wollen wir die Begriffe Internationalisierung und Lokalisierung klären. Diese werden zwar oft synonym verwendet, es handelt sich jedoch um unterschiedliche Schritte bei der Bereitstellung einer Anwendung zur internationalen Nutzung. Durch Internationalisierung – häufig aus dem Englischen mit dem Numeronym i18n abgekürzt – wird eine Anwendung so vorbereitet, dass sie per Konfiguration leicht für verschiedene Sprachen und Länder angepasst werden kann. Hierfür setzt man beispielsweise statt hartkodierter Texte Platzhalter ein und bereitet die Unterstützung verschiedener Zahlen- und Datumsformate vor. Im nächsten Schritt, der Lokalisierung (localization = l10n), werden die Übersetzungen für die jeweiligen Sprachen hinzugefügt, ohne dass weitere Änderungen im Code notwendig sind. Die Sprache und das Land, an die die Software angepasst werden soll, werden über die Locale definiert. Diese kann auch weitere Eigenschaften vorgeben, zum Beispiel den verwendeten Zeichensatz oder die Sortierungen. JavaScript nutzt dabei die Locale-Codes nach dem Standard BCP 47 [1]. Beispiele hierfür sind de, de-DE und de-CH.

Aufgrund des Umfangs des Themas wird dieser Artikel sich auf die folgenden, gängigen i18n-Fragestellungen beschränken:

- Übersetzungen: Ersetzen von Texten in der Anwendung mit Stringinterpolation und Pluralbildung
- Formatierung: Anzeige von Datums-, Zeit- und Zahlenformaten sowie Währungen, abhängig von der Locale des Users
- Parsen: Verarbeitung von Eingaben in einem Locale-abhängigen Datums-, Zeit- und Zahlenformat
- Sortierung: Berücksichtigung von nationalen Regeln bei Textvergleichen

Auf weitere, speziellere Anforderungen – wie zum Beispiel linksläufige Schreibrichtung oder die Verarbeitung von Telefonnummern, Adressen und Postleitzahlen – kann hier nicht näher eingegangen werden. Bezüglich der Schreibrichtung, die nicht per JavaScript, sondern per HTML und CSS gelöst wird, sei an dieser Stelle auf den Artikel „Right-To-Left Development In Mobile Design“ von Robert Dodis [2] verwiesen.

Standards: CLDR und ICU

Zur Internationalisierung und Lokalisierung von Anwendungen gibt es eine breite Palette unterschiedlicher Standards beziehungsweise Standardformate. Zwei Projekte sind hierbei besonders relevant, nämlich das Common Locale Data Repository (CLDR) und die International Components for Unicode (ICU).

Bei CLDR handelt es sich um ein Projekt des Unicode-Konsortiums [3]. Es stellt ein umfangreiches Datenset mit Lokalisierungsinformationen bereit. Enthalten sind unter anderem Übersetzungen für Sprachen, Länder, Zeitzonen und Währungen sowie Datums-, Zeit- und Zahlenformate. Die Daten sind in 200 Sprachen für 740 Locales verfügbar, allerdings in unterschiedlicher Vollständigkeit. Eine Übersicht über die Abdeckung pro Locale wird gepflegt [4]. Neue Versionen des CLDR werden zweimal im Jahr veröffentlicht.

Viele große Unternehmen verwenden die CLDR-Daten, darunter Microsoft, Apple und Google. CLDR ist der De-facto-Standard für Locale-Daten und kommt in verschiedensten Softwareprodukten zum Einsatz. Als Entwickler nutzt man das CLDR meist indirekt über Drittbibliotheken. Auch in Java sind die CLDR-Daten

ab JDK 8 enthalten. Seit JDK 9 ist es das Defaultdatenset für die Lokalisierung in Java.

Bei ICU (International Components for Unicode) handelt es sich um ein Open-Source-Projekt für Unicode-Unterstützung und Internationalisierung [5]. Es wurde von IBM gestartet und 2016 an das Unicode-Konsortium übergeben. ICU stellt eine C-/C++- und Java-Bibliothek (icu4j) bereit, deren Umfang beispielsweise auf Java-Seite über die i18n-Funktionen im JDK hinausgeht. Die Releases der icu4j-Bibliothek beinhalten jeweils das aktuellste CLDR-Datenset und unterstützen die aktuelle Unicode-Version.

ICU definiert ein Messageformat, dessen Syntax neben Platzhaltern auch Formatierung, Pluralbildung und Fallunterscheidungen erlaubt. Platzhalter werden wie beim *java.text.MessageFormat* in geschweiften Klammern geschrieben. Allerdings können die Platzhalter nicht nur durchnummeriert, sondern auch benannt werden. Eine gute Übersicht über das ICU-Messageformat findet sich bei FormatJS [6]. Werden die Möglichkeiten der Messagesyntax in vollem Umfang genutzt, können Übersetzungstexte beziehungsweise -ausdrücke kompliziert werden. In diesem Fall ist ein entsprechender Toolsupport auf Seiten der Übersetzer notwendig. Listing 1 zeigt mehrere Übersetzungstexte aus unserer Demoanwendung im ICU-Messageformat.

i18n-Bordmittel im Browser

Moderne Browser bieten über das JavaScript-*Intl*-Objekt bereits nativ wichtige Internationalisierungsfunktionen. Das *Intl*-Objekt ist durch das JavaScript Internationalization API (ECMA-402) spezifiziert. Im Internet Explorer ab Version 11 sowie in Edge, Chrome und Firefox sind daher folgende Funktionen nutzbar [7]:

- *Intl.DateTimeFormat* zur Datums- und Zeitformatierung
- *Intl.NumberFormat* zur Zahlenformatierung und für Währungen
- *Intl.Collator* zur sprachabhängigen Sortierung von Strings

Listing 2 zeigt Beispiele für die Verwendung des *Intl*-Objekts. Alle Eigenschaften des *Intl*-Objekts wurden detailliert kommentiert [8]. Falls notwendig, können ältere Browser per Polyfill nachgerüstet werden. Die *Intl*-Funktionen lassen sich auch komfortabel per *Date.prototype.toLocaleString*, *Number.prototype.toLocaleString* und *String.prototype.localeCompare* nutzen.

Formatierungs- und Sortierfunktionen sind somit größtenteils über den Browser abgedeckt. Leider gibt es browserübergreifend nicht ausreichend Möglichkeiten, Usereingaben Locale-basiert zu parsen. Das ist insbesondere bei Zahlen- und Datumseingaben notwendig. Hierfür sind im HTML-Standard Inputfelder vom Type *number* und *date* vorgesehen. Bei beiden Feldtypen gibt es allerdings größere Unterschiede zwischen den Browsern. Bei *number*-Inputs akzeptieren Chrome und

Die Intl-Funktionen des Browsers und die i18n-Bibliotheken benötigen die Locale des Users, um die jeweiligen Übersetzungen und weitere Daten anzuzeigen.

Firefox beispielsweise Punkt und Komma als Dezimaltrennzeichen, Edge und IE 11 akzeptieren nur Punkt. Beim *date*-Input gibt es ebenfalls kein durchgängiges Verhalten [9]. Falls eine Anwendung möglichst viele Browser unterstützen soll, müssen daher entsprechende Komponenten auf Basis eines normalen Textinputs eingesetzt werden. Die Verarbeitung von Datumseingaben wird im Normalfall über eine Datepicker-Komponente aus einer UI-Bibliothek gelöst. Zur Verarbeitung von Zahleneingaben kann gegebenenfalls auch eine eigene

Listing 1

```
{
  "example.textWithPlaceholder": "Wir nutzen {technology}.",
  // Wir nutzen React. (technology = "React")

  "example.plural": "TIMOCOM hat {count, plural, one {eine
                Entwicklerstelle} other {# Entwicklerstellen}} zu besetzen.",
  // TIMOCOM hat eine Entwicklerstelle zu besetzen. (count = 1)
  // TIMOCOM hat 10 Entwicklerstellen zu besetzen. (count = 10)

  "example.format": "Die Anwendung ist zu {completionStatus, number,
                                                    percent} fertig!"
  // Die Anwendung ist zu 80 % fertig! (completionStatus = 0.8)
}
```

Listing 2

```
new Intl.DateTimeFormat("de").format(Date.now())
// result: 22.10.2018

new Intl.DateTimeFormat("en", {weekday: "long", year: "numeric",
                    month: "long", day: "numeric"}).format(Date.now())
// result: Monday, October 22, 2018

new Intl.NumberFormat("de").format(9876543.21)
// result: 9.876.543,21

new Intl.NumberFormat("en", {style: "currency", currency: "EUR",
    currencyDisplay: "name", minimumFractionDigits: 0}).format(1000000)
// result: 1,000,000 euros
```

Implementierung ausreichen. Auch für die Behandlung von Übersetzungstexten und beispielsweise für relative Zeitformate muss auf Bibliotheken zurückgegriffen werden. Intl API wird im Rahmen des TC39-Prozesses übrigens kontinuierlich weiterentwickelt [10].

Die *Intl*-Funktionen des Browsers und die i18n-Bibliotheken benötigen die Locale des Users, um die jeweiligen Übersetzungen und weitere Lokalisierungsdaten anzuzeigen. Die Locale kann über Browserbordmittel festgestellt werden. Es gibt mehrere Spracheinstellungen, die abgefragt werden können, unter anderem auch die Systemsprache. Verlässlicher ist allerdings die Liste der durch den User im Browser konfigurierten Sprachen. Diese kann zum einen per JavaScript über *navigator.languages* abgefragt werden. Sie wird vom Browser aber auch als Accept-Language-Header beim Server-Request übertragen. Leider unterstützt IE 11 den Zugriff auf *navigator.languages* nicht. In diesem Fall sollte die Locale per Server-Request ermittelt werden – zum Beispiel beim Laden der HTML-Seite, in die die SPA eingebettet ist. Eine Variante ist, die serverseitig ermittelte Locale in die HTML-Seite einzutragen, sodass der JavaScript-Client auf die Information zugreifen kann.

i18n-Bibliotheken

Im JavaScript-Umfeld gibt es eine Vielzahl von i18n-Bibliotheken. Da sie teilweise unterschiedliche Aspekte der Internationalisierung behandeln, sind sie oft nicht direkt miteinander vergleichbar. Wir betrachten im Folgenden drei besonders verbreitete Bibliotheken – siehe auch Tabelle 1.

Die umfangreichste Internationalisierungsbibliothek ist Globalize [11]. Sie unterstützt das ICU-Messageformat, Zeit-, Datums- und Zahlenformatierung sowie das Parsen von Datums- und Zahleneingaben. Die Bibliothek ist in einzelne Module aufgeteilt. Die Funktionsblöcke wie zum Beispiel Zahlenformatierung und -parsing können somit einzeln mit der Anwendung gebündelt werden. Die Bibliothek beinhaltet – unter anderem aus historischen Gründen – auch die CLDR-Daten, da ihre Entwicklung startete, als der Intl-Standard durch die Browser noch nicht großflächig unterstützt wurde. Das hat bei der Nutzung im Browser den Nachteil, dass der umfangreiche Datensatz für die jeweilige Locale aus der Bibliothek geladen werden muss. Auf der anderen Seite bietet es den Vorteil, dass die Lokalisierung in jedem Fall über alle Browser hinweg konsistent ist. Bei der Nutzung des *Intl*-Objekts ist das zumindest für exotische Locales nicht garantiert, denn der Standard definiert nicht, auf welchen Daten die Browserhersteller aufbauen. Die ECMA-402-Spezifikation empfiehlt für die Implementierung durch die Browserhersteller zwar die Verwendung des CLDR, das CLDR-Datenset inklusive einer genauen Version ist aber nicht festgeschrieben.

Die i18n-Bibliotheken mit den höchsten npm-Downloadzahlen sind FormatJS und das darauf aufbauende React Intl [12], [13]. Bei beiden handelt es sich um Open-Source-Projekte von Yahoo. FormatJS unterstützt das ICU-Messageformat und die Zeit-, Datums- und Zahlenformatierung. Es nutzt das *Intl*-Objekt und bietet für bessere Performance ein Caching der *Intl*-Instanzen. React Intl ist die meistgenutzte i18n Library im React-Umfeld. Wir werden sie später noch genauer vorstellen.

Ein ebenfalls verbreitetes Framework ist i18next [14]. Schwerpunkt von i18next sind Übersetzungen. Das Framework ist über Plug-ins erweiterbar und bietet ein umfangreiches Ökosystem. i18next definiert ein eigenes Messageformat. Das ICU-Messageformat wird über ein Zusatz-Plug-in unterstützt. Andere Erweiterungen bieten unter anderem Funktionen zur Bestimmung der User-Locale sowie zum Laden und Cachen der Übersetzungen. Verschiedene zusätzliche Bibliotheken ermöglichen eine i18next-Integration mit Vue, Angular oder React.

Neben den genannten Projekten kommen zur Verarbeitung und Anzeige von Datumsformaten oftmals spezielle Bibliotheken zum Einsatz. Die populärsten sind: Moment.js, date-fns und Luxon [15]. UI-Komponentenbibliotheken bauen ihre Datepicker meist auf einer dieser Date-Libraries auf. Daher bestimmt die Auswahl der UI-Komponentensammlung häufig auch die Wahl der Datumsbibliothek.

Umgang mit Lokalisierungsdaten

Nachdem wir die Grundlagen für Internationalisierung und Lokalisierung abgesteckt haben, betrachten wir als Nächstes, wie Übersetzungsdateien und weitere Lokalisierungsinformationen in der Anwendung bereitgestellt werden.

	Globalize	FormatJS	i18next
ICU-Messageformat	Ja	Ja	Ja, über Plugin
Formatierung (Zahlen, Datum, Zeit)	Ja	Ja	Nein
Parsen (Zahlen, Datum, Zeit)	Ja	Nein	Nein
Automatisches Extrahieren und Prüfen von Keys	Nein	Ja, per babel-plugin-react-intl	Ja, per i18next-scanner
Erweiterung für Frontend-Bibliotheken	React	React, Ember	React, Vue, Angular, Ember ...
npm-Downloads pro Woche (Stand: 10/2018)	40 000	326 000 (intl-messageformat) 229 000 (react-intl)	168 000 (i18next) 73 000 (react-i18next)

Tabelle 1: JavaScript-i18n-Bibliotheken

Die Übersetzungen sind oftmals als Schlüssel-Wert-Paare im JSON-Format hinterlegt. Die meisten Übersetzungstools unterstützen JSON als Austauschformat zwischen Entwicklern und Übersetzern. Bequemerweise bieten viele i18n-Bibliotheken auch ein Tooling, um die Schlüssel und gegebenenfalls gesetzte Defaultwerte in eine Übersetzungsdatei zu extrahieren. Diese kann im Anschluss zur Übersetzung weitergegeben werden. Über das Tooling kann häufig auch geprüft werden, ob alle übersetzten Dateien die gleichen Keys enthalten. Die JSON-Dateien sind dabei entweder mit flachen Attributen oder als Objektbaum strukturiert (Listing 3). Der Schlüssel ist in beiden Fällen üblicherweise eine Kennung, die sich aus der Struktur der Seite beziehungsweise der Komponenten ergibt.

Es gibt unterschiedliche Verfahren, die Übersetzungen in die JavaScript-Anwendung zu verpacken, sodass diese später – abhängig von der jeweiligen Locale des Users – angezeigt werden können.

Ein erster, simpler Ansatz ist es, alle Übersetzungsdateien direkt mit den Anwendungs-Sourcen zu bündeln – beispielsweise per webpack. Dieses Vorgehen wirkt sich natürlich auf die Größe des Anwendungsartefakts aus. Es kann aber eine valide Option sein, wenn die zusätzlichen sprachabhängigen Daten die Ladeperformance der Applikation nicht entscheidend beeinflussen. Das trifft zu, wenn die Anwendung nur im ohnehin schnellen Unternehmensnetzwerk betrieben wird oder wenn nur wenige Übersetzungstexte beziehungsweise Sprachen bereitgestellt werden. Bei größeren Internetanwendungen ist dieser Ansatz allerdings nicht wünschenswert. Hier geht es meist darum, die Time to Interactive klein zu halten und hierfür das Anwendungspaket möglichst optimiert auszuliefern. In diesem Fall sollten nur die Übersetzungen geladen werden, die unbedingt notwendig sind.

Eine Möglichkeit zur Optimierung ist die statische Lokalisierung zur Build-Zeit. Dieses Verfahren wird beispielsweise durch den i18n-Mechanismus von Angular unterstützt [16]. Das Angular CLI und das Framework stellen dazu spezielle Funktionen zur Verfügung, um Texte aus den HTML-Templates zu extrahieren und auf Basis der Übersetzungsdateien separate Anwendungsartefakte für jede unterstützte Sprache zu erzeugen. Das heißt, die übersetzten Texte werden durch das Tooling direkt in das jeweilige Artefakt kompiliert. Auch bei Nicht-Angular-Anwendungen kann dieses Vorgehen, zumindest für sehr einfache Stringersetzungen, per Webpack-Plug-in durchgeführt werden [17].

Wesentlich gängiger als der statische Ansatz ist die dynamische Lokalisierung der Anwendung. Hier wird die Übersetzungsdatei für die Sprache des Users zur Laufzeit geladen und liegt danach im Browser gecachet vor. Das dynamische Nachladen der Daten kann dabei über verschiedene Wege erfolgen. Die Übersetzungen können per XHR von einem REST Service geladen oder über ein separates *script*-Tag in die initiale HTML-Seite der Anwendung eingebunden werden. Eine zusätzliche Alternative ist der dynamische Import eines ES-Moduls, das die Lokalisierungsdaten enthält. Diese Variante betrachten wir im folgenden Abschnitt genauer.

Lokalisierungsdaten dynamisch importieren

Das ES-Modulsystem unterstützt im aktuellen Standard nur statische Imports. Die Spracherweiterung für dynamische Imports ist aktuell in Proposal Stage 3 und wird wahrscheinlich bis zur nächsten ECMAScript-Version finalisiert. Sie wird aber bereits von allen relevanten Tools – webpack, Babel, TypeScript – unterstützt. Listing 4 zeigt ein dynamisches Importstatement. Da es sich beim dynamischen Import um eine asynchrone Operation handelt, wird ein Promise zurückgegeben.

Für webpack ist der dynamische Import beim Bündeln der Anwendung der Marker für das Codesplitting. Webpack erstellt einen neuen Chunk, das heißt ein neues JS-Bundle, das das dynamisch importierte Modul und dessen Abhängigkeiten enthält. Außerdem stellt webpack die Funktionalität bereit, die für das asynchrone Laden im Hintergrund benötigt wird. Um das Codesplitting anzupassen, können die sogenannten Magic Comments von webpack genutzt werden. Über den *webpackChunkName* lässt sich beispielweise der Name des Bundles konfigurieren. Im gezeigten Code wird hierfür der Platzhalter *[request]* verwendet, wodurch webpack den Namen des Moduls als Bundle-Namen übernimmt.

Benötigt die Anwendung nicht nur Übersetzungen, sondern weitere Lokalisierungen – zum Beispiel für die verwendete Date-Bibliothek – ist es sinnvoll, ein

Listing 3

```
// flat keys
{
  "myComponent.aText": "Chuck Norris doesn't wear a watch.",
  "myComponent.anotherText": "He decides what time it is."
}

// nested
{
  "myComponent": {
    "aText": "Chuck Norris doesn't read books.",
    "anotherText": "He stares them down until he gets the information he
                                                              wants."
  }
}
```

Listing 4

```
loadLocaleData(locale: string): Promise<LocaleData> {
  return import(/* webpackChunkName: "[request]" */ "../localisations/
                                                  LocaleBundle_" + locale);
}
```

Die Kontextinformationen stehen nicht nur für die direkten Kindkomponenten bereit, sondern für den gesamten darunterliegenden Teil des Komponentenbaums.

spezielles Locale-Modul zu erstellen. Dieses importiert statisch die Übersetzungstexte und die jeweiligen Locale-Daten aus den verwendeten Bibliotheken. Dadurch enthält das von webpack generierte Locale Bundle alle zur Lokalisierung benötigten Daten. Listing 5 zeigt das entsprechende deutsche Locale Bundle aus der Beispielanwendung.

In der Demoanwendung ist das Nachladen des Sprach-Bundles ebenfalls implementiert. Zur einfacheren Darstellung lädt die Anwendung zum Start immer die englische Lokalisierung. Über die Oberfläche kann der Anwender die Sprache auf Deutsch wechseln. Daraufhin wird das entsprechende Locale Bundle geladen und die Sprache in der Anwendung umgeschaltet. Über die Web-Developer-Tools des Browsers lässt sich das Laden des Bundles bei der Sprachumschaltung nachvollziehen.

Listing 5

```
import messages from "./messages/messages_de.json";
import reactIntlLocaleData from "react-intl/locale-data/de";

export default {
  locale: 'de',
  messages,
  reactIntlLocaleData
}
```

Internationalisierung einer React-Anwendung

Zur Internationalisierung unserer Beispielanwendung verwenden wir React Intl. Die Bibliothek bietet – auf Basis der Core-Funktionen von FormatJS – React-Komponenten zur Datums- und Zahlenformatierung sowie für die Darstellung von Übersetzungstexten im ICU-Messageformat.

Listing 6 zeigt die Nutzung von React Intl an einem einfachen Beispiel. Am Anfang müssen zwei Initialisierungsschritte erfolgen. Zum einen benötigt die Bibliothek – zur Pluralbildung und für das relative Zeitformat – Lokalisierungsdaten, die nicht per *Intl*-Objekt im Browser bereitstehen. Diese werden im React-Intl-Package pro Sprache angeboten und sind in Listing 5 in unser Locale-Modul importiert worden. Im Anschluss werden sie über die Methode *addLocaleData* an React Intl übergeben.

Als zweiten Schritt müssen wir über die *IntlProvider*-Komponente einen i18n-Kontext erzeugen. Der Kontext in React ist ein Mechanismus, über den Informationen von einer Elternkomponente an Kindkomponenten weitergegeben werden können, ohne dass diese Informationen direkt als Properties (das heißt Komponentenattribute) weitergereicht werden müssen. Die Kontextinformationen stehen nicht nur den direkten Kindkomponenten bereit, sondern sind ab der Elternkomponente (= Provider) im gesamten darunterliegenden Teil des Komponentenbaums zugänglich. Die Verwendung des Kontexts ist insbesondere bei Bibliotheken gängig, da diese keine Annahmen über den Aufbau der jeweiligen Anwendung treffen können. Der

Listing 6

```
import * as React from "react";
import {IntlProvider, FormattedMessage, FormattedNumber,
                    FormattedDate, FormattedRelative} from "react-intl";
import * as ReactIntl from "react-intl";
import localeData from "./localisations/LocaleBundle_de";

ReactIntl.addLocaleData(localeData.reactIntlLocaleData);

export default class MySimpleIntlApp extends React.Component {

  render() {
    return (
      <IntlProvider locale="de" messages={localeData.messages}>
        <>
          <FormattedMessage id="example.textWithPlaceholder"
                                    values={{technology: "react-intl"}}/>
          // renders: <span>Wir nutzen react-intl.</span>

          <FormattedNumber value={1234.56} style="currency"
                                    currency="EUR" currencyDisplay="name"/>
          // renders: <span>1.234,56 Euro</span>

          <FormattedDate value={Date.now()}/>
          // renders: <span>21.10.2018</span>

          <FormattedRelative value={Date.now()}/>
          // renders: <span>vor 2 Minuten</span>
        </>
      </IntlProvider>
    );
  }
}
```

Kontext war längere Zeit ein experimentelles Feature in React. Ab Version 16.3 wurde das Context API überarbeitet und offiziell freigegeben. Der über den *IntlProvider* erzeugte i18n-Kontext wird von React Intl genutzt, um den eigenen Komponenten Informationen – wie zum Beispiel die aktuelle Locale und die Übersetzungstexte – zugänglich zu machen.

Nach den beiden Vorbereitungsschritten können die eigentlichen Formatierungskomponenten genutzt werden. Die Komponente *FormattedMessage* wird zur Anzeige der Übersetzungstexte genutzt. Hier können der Message-Key, die Platzhalterwerte sowie ein Defaulttext übergeben werden. *FormattedNumber* und *FormattedDate* bieten ein komfortables API zum Zugriff auf die Funktionen des *Intl*-Objekts. Die *FormattedRelative*-Komponente ist nützlich, wenn relative Zeitformate benötigt werden, und aktualisiert sich alle zehn Sekunden automatisch. Die Ausprägung der i18n-Funktionen als Komponenten mag zunächst ungewöhnlich erscheinen, entspricht aber der React-Philosophie und bietet im Gegensatz zu einfachen Funktionsaufrufen Vorteile im Hinblick auf die Renderingperformance. Die Komponenten implementieren dazu die *shouldComponentUpdate*-Methode des React-Lebenszyklus. Ein tieferer Einstieg in React und React Intl [18], [13] ist an dieser Stelle nicht möglich.

In der Demoanwendung wird dieses Beispiel weitergeführt, unter anderem wird der Sprachwechsel implementiert. Für diesen wird nach dem dynamischen Import der Locale-Daten erneut die Methode *addLocaleData* aufgerufen und die Properties der *IntlProvider*-Komponente werden aktualisiert. Die Anwendung ist bei GitHub [19] abgelegt. Dort können die Codeänderungen nachvollzogen werden.

Fazit

Wir haben verschiedene Vorgehensweisen und Bibliotheken vorgestellt, um JavaScript-Anwendungen zu internationalisieren. Klar ist, dass es für die Umsetzung wieder einmal nicht den einen Königsweg gibt. Stattdessen gilt es, die geeignete Lösung auszuwählen – den jeweiligen Anforderungen entsprechend.

Der Intl-Standard bietet für die Punkte Formatierung und Sortierung eine gute Basis. Die Verwendung empfiehlt sich auch, um die Anwendungsartefakte klein zu halten. Hierauf aufbauend können weitere Anforderungen über zusätzliche Bibliotheken abgedeckt werden. Für Übersetzungen bietet sich das ICU-Messageformat an, das von vielen Frameworks unterstützt wird.

Die bereitgestellte Demoanwendung zeigt einen Ansatz, um eine React-Anwendung zu internationalisieren. Sie kann als Einstieg in das Thema dienen und als Startpunkt für eigene Experimente verwendet werden.

Philip Stroh arbeitet als Softwarearchitekt bei der TIMOCOM GmbH. Er hat langjährige Erfahrung in der Entwicklung und Konzeption von Webanwendungen mit Java und JavaScript. Seine weiteren Schwerpunkte sind NoSQL-Datenbanken und Delivery-Automatisierung.

Links & Literatur

[1] https://tools.ietf.org/html/rfc5646

[2] https://www.smashingmagazine.com/2017/11/right-to-left-mobile-design/

[3] http://cldr.unicode.org

[4] http://www.unicode.org/cldr/charts/32/supplemental/locale_coverage.html

[5] http://site.icu-project.org

[6] https://formatjs.io/guides/message-syntax/

[7] https://caniuse.com/#feat=internationalization

[8] https://developer.mozilla.org/de/docs/Web/JavaScript/Reference/Global_Objects/Intl

[9] https://caniuse.com/#feat=input-datetime

[10] https://github.com/tc39/proposals/tree/master/ecma402/

[11] https://github.com/globalizejs/globalize/

[12] https://formatjs.io

[13] https://github.com/yahoo/react-intl/

[14] https://www.i18next.com

[15] https://blog.bitsrc.io/9-javascript-date-time-libraries-for-2018-12d82f37872d

[16] https://angular.io/guide/i18n

[17] https://webpack.js.org/plugins/i18n-webpack-plugin/

[18] https://reactjs.org

[19] https://github.com/pstrh/react-i18n-example/