Von nativen Apps zu PWAs

Martin Weissenböck

2020 werden – so eine Prognose – 50% aller Apps als Progressive Web Apps (PWAs) geschrieben werden. Ein wenig googeln nach PWA liefert viele Artikel, die die Vorzüge und Eigenschaften von PWAs beschreiben.

Was ist eine PWA?

Native Apps müssen für jedes Betriebssystem (Android, iOS und was da sonst noch umher krabbelt) extra geschrieben und dann auch gewartet werden. Native Apps werden (gegebenenfalls gegen Gebühr) in diverse Stores gestellt, dann von diesen Stores geladen und installiert. Sie brauchen rasch ein paar MB Platz.

Eine PWA ist im Gegensatz dazu im Prinzip eine Webseite, die mit ECMAScript Funktionen dazu gebracht wird, sich wie eine App zu verhalten und auch genau so am Bildschirm zu erscheinen. Eine Installation ist nicht notwendig, es reicht die Web-Adresse aufzurufen. Trotzdem kann die Seite auf Wunsch über ein Icon (wie eine App) immer wieder gestartet werden.

ECMAScript

Diesem Thema ist ein eigener Beitrag in diesem Heft gewidmet. Aktuelle ECMAScript-Versionen setzen moderne Programm-Paradigmen um. Aber schon zum Lesen von Beispielprogrammen ist zumindest ein grundlegenden Verständnis der Neuerungen notwendig.

Promise

Sprachelemente wie *Template Strings* oder *Arrow Functions* sind leicht zu verstehen. Mit neuen Syntax-Elementen kommen auch weitere neue Konzepte zur Programmablaufsteuerung. Nun treten beim Abarbeiten von Webseiten asynchrone Abläufe auf. Mit Funktionen als Parameter von Funktionen (so genannten *Callback*-Aufrufen) lassen sich zwar diese Abläufen programmieren – die Programme werden aber bei geschachtelten Aufrufen rasch sehr unübersichtlich. Neue Sprachelemente erlauben übersichtlichere Programme: für PWAs ist das *Promise*-Konzept das Wichtigste.

Wozu wurden Promises entwickelt?

Programme waren lange Zeit eine Abfolge von Befehlen, angereichert durch Verzweigungen und Schleifen. Auch Unterprogrammaufrufe (Funktionsaufrufe) passen noch gut in das Schema.

Dann kamen *Interrupts*: Ereignisse, die den Programmablauf zu einem nicht vorhersehbaren Zeitpunkt unterbrechen. Wenn wir aber die *Interrupt-Serviceroutine* als ein Unterprogramm ansehen, nach dessen Beendigung der normale Programmablauf wieder aufgenommen wird, ist das Weltbild immer noch in Ordnung.

Es kamen immer mehr neue Elemente, die den gewohnten Programmablauf durch einander brachten: `try-catch`-Blöcke erlaubten beispielsweise die saubere Behandlung von Fehlern oder sonstigen Ausnahmesituationen.

Die Grundidee der funktionalen Programmierung besteht darin, Funktionen als Objekte zu verwenden und beispielsweise eine Funktion einer Variablen zuzuweisen oder als Argument an eine andere Funktion zu übergeben. Damit war der Weg zur nächsten Erweiterung geöffnet: es wurde ja auch notwendig, per Programm auf Eingaben von Nutzern über die Tastatur, die Maus oder einen Touchscreen reagieren zu können. Die Lösung ist die *Callback*-Funktion.

Wird eine Funktion aufgerufen, werden deren Programmschritte sofort ausgeführt. Mit einer *Callback*-Funktion wird dagegen festgelegt, welche Schritte (erst) beim Eintritt eines Ereignisses ausgeführt werden sollen. Eine typische HTML/Javascript-Kombination sieht etwa so aus:

Wer nicht sicher ist, ob er diesen Beitrag bis zum Ende lesen will, möge einmal einige Webseiten ansehen, die mit PWAs umgesetzt wurden:

https://pwa.rocks/

Interessiert? Dann sehen wir uns doch gemeinsam an, womit sich Programmierer auseinander setzen, die selbst PWAs schreiben wollen. Und dann noch die Prognose mit dem provozierenden Untertitel „Native Apps sind tot – es lebe die Web-App“:

https://www.medienkraft.at/2020-werden-50-prozent-der-apps-bereits-progressive-web-apps-pwas-sein/

```
<button onclick="tuwas">Drück mich</button>
<script> function tuwas() { alert("Taste gedrückt") }; </script>
```

Je komplexer die Aufgaben sind, umso mühsamer wird der Umfang der *Callback*-Funktionen. Die Einführung von *Promise* geht einen Schritt weiter und kommt ohne immer komplizierter werdende *Callback*-Funktionen aus.

Beim herkömmlichen Konzept der synchronen Programme kann in Schritt erst ausgeführt werden, wenn der vorhergehende abgeschlossen ist. Das kann einen Programmablauf sehr verzögern. Asynchrone Programmteile warten nur wenn sie dazu aufgefordert werden auf andere Programmteile. Um einen geregelten Ablauf sicher zu stellen, besteht das Programm aus „Versprechungen“ (`promises`): erst wenn eine versprochene Aktion fertig ist, wird der mit `"then"` angehängte nächste Schritt ausgeführt usw. Geht ein Schritt schief, kann mit dem nächsten `„catch"` darauf reagiert werden. Alle `then`- und `catch`-Befehle enthalten eine (Callback-)Funktion als Parameter: ein weiterer Grund, die Syntax zur Vereinbarung einer (anonymen) Funktion zu vereinfachen.

Das Verständnis dieses Konzepts ist für das Erstellen von PWAs unbedingt notwendig. Eine sehr gute Erklärung ist unter folgenden Links zu finden:

https://developers.google.com/web/fundamentals/primers/promises

https://javascript.info/promise-basics

Es schadet aber auch nicht, „noch weiter vorne“, nämlich hier zu beginnen:

https://developers.google.com/web/fundamentals/web-components/

Cache

Wir erwarten, dass wir überall auf das Internet zugreifen können – sei es über ein WLAN oder über unsere Provider. Und trotzdem gibt es immer wieder Unterbrechungen, zum Beispiel wenn wir mit einem Aufzug fahren. PWAs können dann auf intern gespeicherte Seiten zurück greifen.

Für dieses *Caching* gibt es mehrere Strategien – hier eine kleine Auswahl:

Auf Seiten, die nie oder sehr selten geändert werden, werden die Seiten aus dem internen Speicher blitzschnell geladen. Oder es wird zuerst die interne Seite geladen und, wenn es eine neue gibt, durch diese neue ersetzt. Oder es wird immer aufs Internet zugegriffen und nur, wenn keine Verbindung möglich ist, auf die interne Seite. Wie auch immer – der Zugriff wird extrem beschleunigt.

Single-Page Application (SPA)

Die Kommunikation mit einem Server verläuft (vereinfacht) so:

Eine Webseite wird aufgerufen. Der Server liefert sie. Ein Formular wird ausgefüllt und an den Server geschickt. Der Server antwortet mit einer leicht veränderten oder einer neuen Seite. Aus dem Menü wird eine neue Seite ausgewählt – der Server schickt eine neue usw.

Bei einer SPA oder *Single Page* Webseite werden alle Seiten auf einmal geladen. Ein Teil der Verarbeitung findet schon im Browser statt. Wird eine „neue" Seite angefordert, ist die schon im Speicher zu finden und daher blitzschnell verfügbar. Und noch etwas schneller: die erste Seite wird schon angezeigt, während der Rest noch geladen wird. Wartezeiten sind lästig – es gibt schon viele Möglichkeiten, diese zu verringern.

Natürlich müssen da noch ein paar Dinge im Hintergrund laufen. Wer auf die Zurück-Taste drückt, möchte natürlich wirklich die vorherige Seite der App sehen und nicht die Seite, die zuletzt vom Internet aufgerufen worden ist. Diese Verwaltung erledigt zum Beispiel *Preact*.

DOM – das Document Object Model

Was passiert eigentlich mit dem vom Server gelieferten HTML-Code? Nun, der Browser wandelt die Seite in eine hierarchische Struktur, das *Document Object Model* (kurz *DOM,* https://de.wikipedia.org/wiki/Document_Object_Model), um und erzeugt daraus das Bild, das wir sehen. Wenn nun eine kleine Änderung auf der Seite stattfindet, liefert der Server die komplette Seite neu aus. [Na ja – nicht immer. Mit *jQuery* können erstaunliche Effekte erzielt werden. Aber darauf will ich hier nicht eingehen.] Ja, einiges kann schon im Browser zwischen gespeichert werden, aber die Grundidee bleibt: die Seite muss wieder geladen werden.

Programmbibliotheken, wie *React* oder *Preact*, gehen einen anderen Weg: sie merken sich das Modell, registrieren nur mehr die Differenzen zu einer neuen Seite, werten sie aus und können daher die Seite sehr rasch wieder darstellen. Derartige Programme sind nicht unbedingt für PWAs notwendig, beschleunigen aber den Aufbau der Seiten wesentlich und arbeiten mit PWAs sehr gut zusammen.

Und wenn nun der Client selbst die notwendigen Änderungen berechnet, werden gar keine Daten vom Server übertragen. Damit kann die Seite noch schneller aufgebaut werden!

React

React (https://reactjs.org/) ist eine ECMAScript-Programmsammlung zum Erzeugen von Webseiten, vor allem über die Manipulation des DOMs. Aber *React* kann noch mehr: damit PWAs wirklich mit nativen Apps konkurrieren können, muss auch der Zugriff auf die Hardware möglich sein. Über Plugins für *React* können beispielsweise die Kamera, der GPS-Empfänger, der Fingerabdruckleser oder der Vibrationsalarm angesprochen werden. Die Liste der Komponenten, die damit auch für ECMAScript zur Verfügung stehen, wächst rasch.

Preact

Preact (https://preactjs.com/) ist eine ähnliche Bibliothek, die (größtenteils) zu *React* kompatibel ist, aber beim Speicherplatz, bei der Geschwindigkeit und beim Schreiben des Codes zum Teil recht eindrucksvolle Verbesserungen bringt.

Um mit Hardware-Komponenten, wie etwa der Kamera oder dem GPS-Empfänger, arbeiten zu können, muss oft auf die *React*-Programmbibliothek zurück gegriffen werden. Dank des *React*-Kompatibilitätsmodus ist das aber kein großes Problem.

Mein Eindruck: die Beschäftigung mit dieser Bibliothek bringt die größten Vorteile.

Service Worker

Auch mit den bisher genannten Eigenschaften hätten PWAs nur geringe Chancen, mit nativen Apps zu konkurrieren. Was wir benötigen, ist ein Werkzeug, um Nachrichten auf dem Endgerät empfangen zu können. Und dieser Mechanismus muss auch dann funktionieren, wenn die zu Grunde liegende Webseite nicht geöffnet ist. Wir brauchen also ein Stück Software, das auf *Ereignisse* reagieren kann. Da der Zeitpunkt, wann ein Ereignis auftritt, nicht vorhersehbar ist, sprechen wir von einer asynchronen Bearbeitung.

Ein *Service Worker* erledigt diese Aufgaben, lauscht auf Ereignisse und reagiert darauf. Welche Ereignisse können das sein? Zum Beispiel eine ankommende Nachrichten (zum Thema *Push* und *Notification* gleich mehr). Oder eine abgehende Mitteilung konnte (wegen eine fehlenden Internetverbindung) nicht gesendet werden: dann kann der *Service Worker* selbstständig beim Wiederkehren der Verbindung die Übertragung fortsetzen und abschließen. *Service Worker* setzen PWAs nicht voraus, ergänzen diese aber und schaffen so erst vollwertige Apps.

SSL-gesicherte Übertragung

Auf dem lokalen Server (http://127.0.0.1) kann man das alles leicht ausprobieren. Wird aber die App ins weite Internet entlassen, muss eine sichere Verbindung (SSL-gesichert, also über das https-Protokoll) verwendet werden. Vor einiger Zeit war das eine (für private Experimente) relativ teure Angelegenheit. CAcert ist eine Organisation von Freiwilligen, die die Ausgabe von kostenlosen Zertifikaten zum Ziel hat. Aber CAcert-Zertifikate werden von Browsern nicht direkt unterstützt.

Abhilfe schafft *„Let's encrypt"* (https://letsencrypt.org/). Die Zertifikate sind kostenlos, leicht zu installieren und werden von allen Browsern erkannt. Damit gibt es keinen Grund mehr, für eigene Webseiten keine SSL-Verschlüsselung zu verwenden. *Let's encrypt* ist sehr zu empfehlen.

Ich meine, dass diese Initiative mit Spenden unterstützt werden sollte. Auch SCHUL.InfoSMS verwendet Let's encrypt.

Clientseitige Datenbank

Wie werden Daten über die Lebensdauer eines Aufrufs hinaus aufbewahrt? Jeder Browser verfügt über einen *Cache* – einen „Zwischenspeicher". Der *Cache* ist eine Möglichkeit, aber nicht besonders komfortabel. Zusätzlich stehen aber mehrere Datenbanken zur Verfügung, zum Beispiel die *IndexedDB*: jedes gespeicherte Objekt hat einen Schlüssel *(Key)* und ein Datenfeld oder mehrere Datenfelder.

Nur zur Klarstellung: jeder moderne Browser bringt diese Datenbank schon mit, eine zusätzliche Installation ist nicht notwendig.

Push und Notification

Mit *Push* kann ein Server eine Nachricht an einen Client senden. Wichtig: diese Nachrichten werden vom Absender verschlüsselt und dann übermittelt, kein Browser-Hersteller hat darauf Zugriff! *Push*-Nachrichten können unterschiedliche Inhalte haben, zum Beispiel einen Datenbestand im Hintergrund (also ohne Eingriff des Benutzers) aktualisieren. Wenn eine *Push*-Nachricht am Bildschirm angezeigt werden soll, wird daraus eine *Notification*. Aber auch ohne *Notification* wird der Nutzer über die angekommene Push-Nachricht informiert.

Zum Ausprobieren: https://developers.google.com/web/fundamentals/codelabs/push-notifications/ ist ein kurzer Kurs zum Erstellen von Programmen, die einen *Service Worker* einrichten, *Push*-Nachrichten empfangen und als *Notification* anzeigen.

Für *Push*-Nachrichten und *Notifications* gilt: *der Benutzer muss dem Empfang dieser Nachrichten zustimmen!* Spam wird auf diese Weise verhindert. Wenn ein Benutzer den Empfang von Nachrichten aber *einmal* ablehnt, ist nicht einmal mehr eine nochmalige Einladung (durch den Server) möglich. Nur wenn der Nutzer in den Einstellungen seines Browsers die Sperre aufhebt, kann der Server *Push*-Nachrichten senden.

Push-Nachrichten und *Notifications* werden von *(P)react* sehr gut unterstützt.

Die Einführung der *Push*-Nachrichten wird als eine der *wesentlichsten Innovation der Webtechnologie* in den letzten Jahren bewertet.

Noch ein paar Gedanken zum Thema Datenschutz: Aktuelle Browser-Versionen empfangen Push-Nachrichten verschlüsselt.

Das Zauberwort dafür ist *VAPID („Voluntary Application Server Identification for Web Push“)*. Jeder Browser-Anbieter, der *Push*-Nachrichten in dieser Form anbietet, stellt auch einen eigenen *Messaging*-Dienst zur Verfügung. Damit werden – wie erwähnt – Daten am *eigenen Server* verschlüsselt und *erst am Client* entschlüsselt: sehr sicher und vom Datenschutz her bestens geeignet.

Ein älteres Verfahren heißt *GCM („Google Cloud Messaging“)*, wird per 11. April 2019 eingestellt und von *FCM („Firebase Cloud Messaging“)* abgelöst (https://www.heise.de/developer/meldung/Google-Cloud-Messaging-Abschaltung-am-11-April-2019-4021944.html). Hier führt der Weg der Mitteilung über die Google Server. Google ist eine US-amerikanische Firma und unterliegt dem *Patriot Act*. Ob dabei die Geheimhaltung (Verschlüsselung) lückenlos garantiert wird, darf zumindest hinterfragt werden. Siehe auch https://security.googleblog.com/2018/06/end-to-end-encryption-for-push.html Leider verwenden sehr viele Apps GCM bzw. FCM.

Background Sync

Nachrichten und Daten, die von einem Client (einem Handy) gesendet werden, müssen verlässlich an den Empfänger übermittelt werden. Unabhängig davon, ob wir gerade in einem Versorgungsloch im Waldviertel unterwegs sind, in einen Aufzug einsteigen, eine Tiefgarage aufsuchen oder das WLAN in einem Gebäudeteil nicht funktioniert: sobald die Internetverbindung wieder hergestellt ist, sind die Daten- ohne nochmalige Eingabe oder sonstige Aktionen – zu übertragen.

Auch diese Aufgabe übernimmt der Service Worker. Die Nachricht wird als *Sync Event* dem Service Worker übergeben, der sich verlässlich um die Ausführung kümmert. Na ja – abdrehen darf man das Handy eben nicht...

Node, npm

ECMAScript ist nicht mehr auf den Client (den Browser) beschränkt, sondern wird auch auf der Serverseite eingesetzt. Wichtig ist dabei eine effiziente Implementierung, da ein Server ja viele Anfragen gleichzeitig behandeln muss. Derartig große Programmpakete müssen effizient gewartet werden: Versionen und gegenseitige Abhängigkeiten sind automatisch zu verwalten.

Node ist die Plattform, auf der solche Programm gesammelt werden (https://nodejs.org/en/, https://de.wikipedia.org/wiki/Node.js). Um nun einzelne Programm auf einem Gerät zu installieren, wird *npm* (früher: *Node Package Manager*) (https://www.npmjs.com/, https://de.wikipedia.org/wiki/Npm_(Software)) eingesetzt.

Icons am Startschirm

Eine Kleinigkeit fehlt noch für das *„App Feeling“:* ein Icon am Start-Bildschirm. Aber auch hier sorgen PWAs vor: über eine Manifest-Datei wird festgelegt, was wie angezeigt werden soll. Nach dem ersten Aufruf der PWA wird der Nutzer gefragt, ob er ein Icon installieren will. Mit der positiven Bestätigung ist alles erledigt.

Und was ist daran Progressive?

PWAs können und sollen so geschrieben werden, dass auch ältere Browser eine Nutzung der Webseite ermöglichen. Die Anzeige kann einen geringerem Leistungsumfang oder geringere Geschwindigkeit haben, darf aber nicht ausfallen. Diese Eigenschaft wird als *progressive* bezeichnet.

Das war's jetzt – oder?

Jein! Ja, wer alles brav durchgearbeitet hat, kann schöne PWAs schreiben. Mit der *IndexedDB* bleiben Daten auch wohl geordnet am Mobiltelefon (Tablet, Desktop,...) erhalten. Was aber, wenn ich möchte, dass diese Daten auch auf anderen Geräten von mir oder von anderen verfügbar sein sollen? Und wenn Änderungen an einem Datensatz auch sofort an alle anderen Nutzer dieser Datenbank weiter gegeben werden sollen, also automatisch repliziert werden sollen. Geht das?

Ja! *PouchDB* ist ein Programmpaket, das auf die *IndexedDB* „draufgesetzt“ wird, und selbstständig mit anderen kommuniziert. Die Variante für den Server heißt *CouchDB*. Damit können auch Berechnungen, die am Server durchgeführt werden, schnell an alle Nutzer verteilt werden. Und das geht auch in der umgekehrten Richtung.

CouchDB und PouchDB sind nicht die einzigen Datenbankprogramme für diesen Zwecke – mir erscheint diese Kombination aber besonders effizient.

NoSQL

Viele kennen und arbeiten mit SQL, der *Structured Query Language*. *CouchDB* verwendet aber *NoSQL* (früher als *„No SQL“*, jetzt als *„Not only SQL“* interpretiert). *CouchDB* wird von Apache bereit gestellt. Wer sich noch nie mit *NoSQL* beschäftigt hat, betritt eine völlig neue Welt. So ziemlich alles, was wir über relationale Datenbanken wissen, ist bei *NoSQL* und damit beim Arbeiten mit *CouchDB* zu vergessen. Aber der Ansatz ist faszinierend und der Lohn ist eine Datenbank mit extrem schnellen Abfragen.

Lesen.

ECMAScript, PWAs, DOM, Preact, npm, Node, Cache, Clientseitige Datenbanken, *Service Worker, Push, Notification, Sync, NoSQL.* das wären so die wichtigsten Kapitel, die ich mir für ein Buch wünschen würde. Mit ausführlichen Beispielen, natürlich auch als Source-Code im Internet. Und vielleicht auf Deutsch? Also die eierlegende Wollmilch-Sau für das Erstellen von PWAs. Aber die gibt es leider nicht. Die Anzahl der Bücher zu einzelnen Themen ist überschaubar, zu anderen wieder sehr redundant. Dafür gibt es viele Artikel im Internet, Kommentare, Ratschläge. Dabei stellt sich heraus, dass viele Artikel einander ähneln.

Noch ein Problem: die verwendeten Programmbibliotheken werden weiter entwickelt. In vielen Beträgen fehlt aber eine Angabe zur Version oder einfach ein Datum. Die Suche nach einem Fehler wird nicht einfacher, wenn man auf einen Artikel über eine alte Programmversion zurück greift.

Viele Beiträge stammen von zwei Quellen ab, nicht wenige sind nur abgeschrieben: MDN und Google

Anstelle einer seitenlangen Literaturliste, mit der aus Zeitgründen erst wieder niemand etwas anfangen kann, ist die kurze Liste im Anhang mein Vorschlag, wie ich eine Annäherung an des Thema PWA empfehle.

Zusammenfassung

„Na gut, dann schreibe ich eben jetzt ein paar PWAs.“ Nein, leider, so einfach geht das nicht. In den vorherigen Abschnitten sind Konzepte beschreiben, die alle – zumindest in den Grundzügen – verstanden werden müssen, bevor irgendetwas, das einer PWA ähnlich sieht, entsteht. Es ist zweifellos sehr hilfreich, Beispielprogramme nachzuvollziehen. Aber jeder, der das schon probiert hat, kennt den Effekt: kaum baut man ein paar eigene Ideen ein, gibt es völlig unverständliche Fehlermeldungen, die erst nach langen Recherchen im Internet zu klären sind. Oft folgen dann noch Hinweise, die für den Wissenden völlig klar sind, aber beim Anfänger weitere Recherchen auslösen.

Diese Zeilen sollten aber niemand entmutigen, sondern eher anspornen, nicht aufzugeben. Wenn 2020 schon 50% aller Apps als PWAs geschrieben werden, sollten wir nicht daran vorbei gehen.

Neue Techniken lassen sich in Kursen oft wesentlich schneller erlernen als durch das Studium von Büchern und Webseiten. Viele Leser der PCNEWS sind an Schulen tätig. Die Pädagogischen Hochschulen wären gut beraten, Seminare zum Thema PWA anzubieten.

PWAs und SCHUL.InfoSMS bzw. SCHUL.InfoService

PWAs funktionieren ohne die Installation einer App, brauchen wenig Ressourcen, sind sehr schnell und auch einfach zu bedienen und daher optimal für alle bisherigen und zukünftigen Aufgaben von SCHUL.InfoSMS und SCHUL.InfoService geeignet. Mehr dazu im nächsten Heft der PCNEWS.

PWA-Literatur

Ich nenne die Bücher in der Reihenfolge, wie sie meiner Ansicht nach am schnellsten in die Materie einführen. Anstatt Bücher zu kaufen, können auch Artikel im Internet gelesen werden. Der Vorteil: oft werden Beispielprogramme angeboten, die dann auch sofort ausprobiert werden können. Viele Beträge sind auf Englisch.

Die *Codelabs* von Google sind gut aufbereitet. Allerdings versucht Google, sein Produkt *Firebase* in den Vordergrund zustellen. *Firebase* ist nur für kleinere Projekte kostenlos. Weitere Bedenken: wo die Daten gespeichert sind, ist nicht klar – und schon sind wir mitten in der Problematik der Datenschutzgrundverordnung (siehe weiter unten).

https://codelabs.developers.google.com/

Ein neutrales Angebot kommt von der Mozialla Foundation:

https://developer.mozilla.org/de/

Zum Thema PWA:

https://developer.mozilla.org/en-US/docs/Web/Progressive_web_apps

Aber nun zurück zur Buchliste:

Progressive Web Apps

(Marcel Peters. ISBN 978 1521881644. Deutsch)

Das Buch ist gut geeignet, das Interesse an PWAs zu wecken. Der Preis ist nicht sehr hoch (13,95 € als Taschenbuch, 7,99 € als Kindle-E-Book). Wer sich nicht sicher ist, ob er in diese Materie einsteigen will, geht damit kein großes Risiko ein.

Das Buch verzichtet leider auf *Preact* und aktuelles *ECMAScript*. Die Beispiele können ohne großen Aufwand und mit Erfolg nachvollzogen werden.

Es gibt ein Literaturverzeichnis. Allerdings könnte durchaus prägnanter hervor gehoben werden, dass das Buch über weitere Strecken die Inhalte der entsprechenden Google-Kurse samt Abbildungen (jedoch auf Deutsch übersetzt) wiedergibt. Apropos Abbildungen: das Layout verträgt durchaus eine Verbesserung. Wenn vier Screenshots in der Abbildung 3 dargestellt werden, trägt ein kleiner Abstand oder Rand dazwischen zur Übersichtlichkeit bei. Und das gilt auch für andere Abbildungen.

Building Progressive Web Apps

(Tal Ater. ISBN 978 1491961650. Englisch. 33,82 €)

Am Beispiel einer Webseite für das fiktive „*Gotham Hotel*“ wird Schritt für Schritt gezeigt, wie PWAs zu implementieren sind.

Positiv:

Nach jedem Kapitel liegt eine funktionsfähige App vor, das dann im folgenden Kapitel erweitert wird. Und wenn einmal etwas schiefgeht, steht der korrekte Programmcode für jedes Kapitel auf github.com bereit.

Alle Bestandteile einer PWA werden sehr detailliert erklärt. Während andere Bücher oft ein fertiges Programm vorsetzen, das dann Zeile für Zeile analysiert wird, werden hier neue Elemente zuerst in ihrer Grundstruktur erklärt und dann immer weiter ausgebaut und gleichzeitig verfeinert. Diese Vorgangsweise führt zu einem wesentlich besseren Verständnis. Ein Beispiel: die Verwendung lokaler Datenbanken ist für PWAs sehr wichtig. Dem Umgang mit der Datenbank *IndexedDB* werden hier 42 Seiten gewidmet: Beispiele werden genau erklärt und immer weiter ausgebaut.

Wer das Buch genau durcharbeitet, bekommt eine sehr gute Vorstellung, wie PWAs aufgebaut sind. Besonderer Wert wird auch darauf gelegt, genau zu erklären, wie die wichtigen Komponenten zusammen arbeiten.

Negativ:

Neue *ECMAScript*-Versionen erlauben wesentlich kompaktere Schreibweisen, die aber nicht verwendet werden. Dadurch wird der Programmcode – vor allem bei *Callbacks* – rasch unübersichtlich.

Preact erfordert etwa 4kB an Speicher – eine wirklich kleine Bibliothek. Im Kapitel 6 taucht nun – quasi ohne Vorwarnung – die Verwendung von *jQuery* auf. Das ist ungewöhnlich, da eine *jQuery*-Bibliothek den Programmcode kräftig aufbläht. *Na ja, mit der jQuery-Alternative zepto.js sind es auch nur 9,6KB – aber da geht es ums Prinzip!*

Mobiltelefone bieten viele Zusatzfunktionen: eine eingebaute Kamera, einen GPS-Empfänger, einen Lage- und Beschleunigungssensor usw. Diese Komponenten waren vorerst nur für native Apps zugänglich. Jetzt erlauben *Preact*- und *React*-Programmbibliotheken den Zugriff auf diese Hardware und bringen PWAs auch sehr nahe an native Apps heran. Bedauerlich, dass dies im Buch nicht erwähnt wird.

Für größere Projekte sind moderne Programmansätze, wie etwa Klassen für Komponenten, sehr nützlich und wichtig. Da aber *Preact* und *React* nicht verwendet werden, gibt es auch keine Erklärungen dazu.

Andererseits verbergen *Preact* und *React* die Details der Abläufe – aus dieser Sicht ist der Einstieg ohne Bibliotheken durchaus lehrreich.

Progressive Web Apps with React

(Scott Domes. ISBN 978 1788297554. Englisch. Taschenbuch 25,91 €)

Das Projekt *Chatastrophe* wird Schritt für Schritt auf- und ausgebaut. Auch hier steht der Sourcecode im Internet zur Verfügung.

Positiv:

Aktuelle *ECMAScript*-Versionen werden verwendet.

Sehr nützlich sind auch die Hinweise, wie Programmflüsse analysiert und die Programme optimiert werden können.

Negativ:

Ab dem Kapitel 4 wird *Firebase* verwendet. Für Versuche und kleine Applikationen stellt Google *Firebase* kostenlos zur Verfügung. Bei intensiverer Nutzung fallen Kosten an, deren Höhe nicht genau vorher gesagt werden können. Außerdem ist nicht nachvollziehbar, wo Google die *Firebase*-Daten speichert und ob dabei die europäischen Datenschutzbestimmungen eingehalten werden. Es wäre gut, wenn zusätzlich gezeigt wird, wie das Projekt ohne *Firebase* verwirklicht werden kann.

Preact bietet (praktisch) alle Möglichkeiten von *React*, braucht aber viel weniger Speicherplatz. Zu *React* gibt es eine Unmenge an Literatur, zu *Preact* viel weniger. Ist es daher notwendig, zuerst *React* zu lernen?

Progressive Web-Apps

(Manfred Steyer. ISBN 978 3868027532. Deutsch, E-Book. 2,99 €)

Das E-Book kostet nur 2,99 € und ist auf Deutsch. Es liefert einen schnellen Überblick über die Prinzipien der PWAs. Zur Realisierung eigener Projekte reicht die Beschreibung aber nicht aus – ich nenne es daher nur als Ergänzung.

Außerdem sollte schon im Titel darauf hingewiesen werden, dass *Angular-2* verwendet wird. *Angular* ist wie *React* ein *Javascript Framework* für eigene Programmentwicklungen. Dieser Artikel beschäftigt sich aber mit dem – aus meiner Sicht moderneren und einfacher zu lernenden – *Preact*. Der Aufwand, *Angular-2* zu erlernen, ist nicht zu unterschätzen.

Progressive Web Apps With Preact

(Vu Tran u.a. ISBN 978 1939902535. Englisch, Kindle E-Book 16,99 €)

Ein anderer Zugang zu dem Thema und ein guter Überblick über die wichtigsten Teile von *Preact*. An einigen Stellen wird auf den Unterschied zu *React* hingewiesen: für *React*-Kenner nützlich, für alle, die erst in die Materie einsteigen, nicht sehr hilfreich. Jedenfalls wird hervorgehoben, warum *Preact* dem speicherintensiveren *React* vorzuziehen ist.

Auch in diesem Buch wird wird an einem Beispiel – einer Webseite für *Hacker-News* – gezeigt, wie ein PWA entsteht. Alle Beschreibungen sind sehr detailliert – für den Einstieg eventuell zu detailliert. Schön wäre es, wenn mit jedem Kapitel auch schon ein lauffähiges Programm entwickelt würde. Da ist aber nicht der Fall und es bedarf einiger Geduld, bis man als Leser Dinge auch ausprobieren kann.

An Effective Guide to Modern JavaScript

ECMAScript 2017 / ES8

(Chong Lip Phang. ISBN 978 1974207923. Englisch. 12,31 € bei Amazon)

Ja, *JavaScript* wird ständig weiter entwickelt – nicht nur in der Namensänderung zu *ECMAScript*: Klassen, Objekte, Lamda-Funktionen, Template-Strings. Die neuen Konzepte erlauben das Schreiben von übersichtlichen Programmen. *Trotzdem hat es ECMAScript 8 auch nicht geschafft, eine vernünftige Programmbibliothek für das Formatieren von Texten, Zahlen und Datumsangaben bereit zu stellen. Vielleicht sollten die ECMAScript-Entwickler einmal die entsprechenden Routinen von Python ein wenig näher ansehen?*

Das ist aber eine Kritik an *ECMAScript* und nicht an diesem Buch. Wer nicht die x-te Version von Büchern, die alle mit dem "Hello World"-Programm beginnen, lesen will, sondern die Konzepte (samt Beispielen!) übersichtlich geordnet lesen oder nachschlagen möchte, sollte dieses kleine Handbuch kaufen. Das ist kein Buch zum Einstieg in *ECMAScript* – grundlegende *ECMA(Java)Script*-Kenntnisse sind vorteilhaft. Sehr hilfreich, wenn man die neuen Konzepte kennen lernen oder verwenden will. Auch für das Lesen PWA-Büchern ist das Verständnis der Konstrukte wichtig.

ECMAScript

Ja, aus *JavaScript* wurde *ECMAScript*. Weitere Namen und Versionen siehe https://en.wikipedia.org/wiki/ECMAScript.

Und nicht nur die Bezeichnung hat sich geändert: neue Konzepte wurden hinzugefügt. Hier eine kleine Auswahl neuer Sprachelemente, die für das Arbeiten mit PWAs hilfreich sind und beim Lesen von Programmbeispielen für das Verständnis wichtig sind.

PWAs können auch in „älteren" Sprachversionen geschrieben werden; mit der moderneren Syntax werden die Programme leichter lesbar und übersichtlicher.

Vereinbaren von Variablen

Wird eine Variable mit (beispielsweise)

```
i = 3;
```

vereinbart und initialisiert, ist das eine globale Variable. Solange JavaScript-Programme schmucke 10-Zeiler waren, war das kein Problem. Aber: globale Variablen sind ein schlechter Programmierstil. Besser wäre *innerhalb* einer Funktion:

```
var i = 3;
```

Damit bleibt der Gültigkeitsbereich der Variablen auf die umschließende Funktion beschränkt. Mit

```
let i = 3;
```

wird (vereinfacht ausgedrückt) der Gültigkeitsbereich auf die umschließende Funktion oder den umschließenden Block beschränkt. Daher ist `let` für die Vereinbarung der Laufvariablen einer Schleife vorzuziehen.

Schleifen

Nehmen wir an, wir haben in einer Liste die Temperaturmittelwerte von Wien über 12 Monate aus 2018 gespeichert :

```
var t = [4.2, -0.7, 3.5, 15.8,
        18.7, 21.3, 22.7, 23.4,
        17.4, 13.0, 6.6, 3.0];
```

Oder auch mit

```
const t = [4.2, -0.7, 3.5, 15.8,
          18.7, 21.3, 22.7, 23.4,
          17.4, 13.0, 6.6, 3.0];
```

Die Berechnung des Mittelwertes ist mit

```
var mw = 0;
for (let i=0; i<t.length; i++)
   mw += t[i];
mw /= 12;
```

möglich, aber nicht sehr hübsch zu lesen. Nehmen wir an, dass das Jahr 12 Monate hat: ist seit 45 v.Chr. im julianischen und im gregorianischen Kalender üblich. Dann könnten wir statt `t.length` einfach 12 schreiben. (Wenn auch gestandene C-Programmierer nichts dabei finden … aber ich denke an alle, die das gerade einmal lernen müssen.)

Die `for`-Schleife samt Addition kann auch als

```
for (let i in t)
   mw += t[i];
```

geschrieben werden. Schon besser – schließlich weiß das Programm ja, wie groß das *Array* `t` ist.

Mit ECMAScript 6 ist die – meiner Ansicht nach – übersichtlichste Schreibweise

```
for (let v of t)
   mw += v;
```

möglich. Mit `of` wird nicht über die Indizes, sondern über die Werte (genauer: über die Namen der Eigenschaften) iteriert. Das ist es, was wir eigentlich haben wollten: *„Nimm einzeln jeden Wert v von t und addiere ihn zu mw."*)

Werden aber die Temperaturwerte als Objekt

```
const t = {jaenner:4.2, februar:-0.7,
   maerz:3.5, april:15.8, mai:18.7, :
   juni:21.3, juli:22.7, august:23.4,
   september:17.4, oktober:13.0,
   november:6.6, dezember:3.0};
```

gespeichert, ist trotzdem nur mehr die Variante

```
for (let i in t)
   mw += t[i];
```

möglich.

Zeichenketten

Wir wollen den errechneten Mittelwert samt Mittelwert in einem Satz ausgeben und speichern diesen Satz vorerst in einer Zeichenkette `mwtext`:

```
var mw = 12.4;
var mwtext =
   'Die mittlere Temperatur in Wien war 2018 '
   +mw+' Grad Celsius.';
```

Wenn da mehrere Variablenwerte und Texte gemischt werden, ist das keine sehr elegante Schreibweise. *ECMAScript* 6 führt *Template Strings* ein. Damit wird dieses Programmstück leichter zu schreiben und zu lesen:

```
var mw = 12.4;
var mwtext =
  `Die mittlere Temperatur in Wien war 2018 ${mw}
Grad Celsius.`;
```

Der String ist in diesem Fall durch Gravis (auch *„Backticks"*, Unicode *GRAVE ACCENT*U+0060) eingeschlossen. Statt einer Variablen kann in den geschwungenen Klammern auch ein Ausdruck stehen.

arrow functions

Mit ES6 wurde eine neue Syntax für die Vereinbarung von Funktionen zusätzlich eingeführt. Statt

```
var addiere = function(x, y) {
   return x + y;
};
```

ist nun auch

```
const addiere = (x, y) => { return x + y };
```

oder noch kürzer

```
const addiere = (x, y) => x + y;
```

zulässig. Sehr nützlich in anonymen Funktionen!

Anonyme Funktionen

Programme zur Auswertung von Webseiten müssen auf *Ereignisse (events)* reagieren: Maus-Klicks, Dateneingaben, Berührungen am Schirm usw. Daher wird in Programmen festgelegt, welche Funktion nach dem Eintreten eines Ereignisses aufgerufen wird. Diese Funktionen werden *Callback*-Funktionen genannt.

Hier ein einfaches Beispiel (ohne die Verwendung von Ereignissen):

Die Funktion `ersterSchritt` gibt den Text *"Erster Schritt"* aus. Erst wenn diese Aktion erfolgreich war, wird die Funktion `callback` aufgerufen, die ihrerseits über alert den Text *"Zweiter Schritt"* ausgibt.

```
function ersterSchritt(callback) {
   alert("Erster Schritt");
   callback();
}
function zweiterSchritt() {
  alert("Zweiter Schritt");
}
// Aufruf:
ersterSchritt(zweiterSchritt);
```

Es spricht nichts dagegen, benannte Funktionen (hier: `zweiterSchritt`) zu verwenden. Da aber diese Funktionen oft nur einmal verwendet werden, werden Programme mit anonymen Funktionen kürzer…

```
function ersterSchritt(callback) {
   alert("Erster Schritt");
   callback();
}

// Aufruf:
ersterSchritt(function() {
   alert("Zweiter Schritt");
});
```

… aber nicht unbedingt übersichtlicher. Vor allem, wenn Funktionen geschachtelt werden. Mit *arrow-functions* wird das Ganze in *ECMAScript* zu

```
function ersterSchritt(callback) {
   alert("Erster Schritt");
   callback();
}

// Aufruf:
ersterSchritt(() => {
   alert("Zweiter Schritt");
});
```

Objekte

ECMAScript-Objekte enthalten Paare von *Namen* und *Eigenschaften (properties)* oder *Namen* und *Methoden (methods)*, getrennt durch einen Doppelpunkt und eingeschlossen in geschwungenen Klammern.

```
var temperaturen = {
  ort: "Wien",
  jahr: 2018,
  t: [4.2, -0.7, 3.5, 15.8, 18.7, 21.3,
      22.7, 23.4, .4, 13.0, 6.6, 3.0],
  mittelwert: t => (t.reduce((sum,v)=>
        (sum + v)) / t.length)
}
```

Der Name in einem Paar kann auch variabel sein. Beispiel:

```
const ZEIT = "jahr";
var temperaturen = {
   ort: "Wien",
   [ZEIT]: 2018,
...
}
```

`ort`, `jahr` und `t` sind *Eigenschaften*, `mittelwert` ist eine *Methode*. Das Beispiel zeigt auch, dass Objekte verschiedene Datentypen enthalten können.

Wie gesagt – das ist eine kleine Auswahl von interessanten Sprachelementen der neuen *ECMAScript*-Versionen.

Links

https://medium.freecodecamp.org/write-less-do-more-with-javascript-es6-5fd4a8e50ee2