Gymnasium zum kurfürstlichen Schloss zu Mainz

Facharbeit

Entwicklung einer WebApp

von einer herkömmlichen Website zu einer progressiven WebApp

von
Yannick Félix
12INF2 - Katharina Gwinner

13. Januar 2017 - 24. April 2017

1 Kurzfassung

WebApps wurden von den meisten Benutzern, wenn auch unbewusst, benutzt. Oft fallen diese garnicht auf, da sie intuitiv zu benutzen sind.

Immer dann, wenn wir eine Webseite benutzten können, ohne dabei die Webseite an sich zu verlassen, oder neuzuladen, handelt es sich um eine WebApp. Einige Beispiele wären die Google Suche, Google's Email Dienst GMail oder auch Facebook.

Große Firmen stecken oft viel Geld und Ressourcen in die Entwicklung von nativen Apps für die beliebtesten Betriebssysteme, darunter Windows, Android und iOS. WebApps und auch deren Erweiterung progressive WebApps sind jedoch plattformunabhängig und sollen in Zukunft auf jedem Betriebssystem benutzt werden können. Somit genügt es eine Anwendung einmal als (progressive) WebApp zu entwickeln.

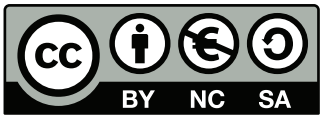
Eine progressive WebApp ermöglicht es sogar den Eindruck zu erwecken, dass sie eine native App sei, ohne dabei auf bekannte Features, wie einen Offline-Modus und Push-Benachrichtigungen zu verzichten, obwohl sie in Wahrheit eine eher normale Webseite darstellt.

Einige Nachteile sind zu Zeit die fehlende Unterstützung unter Apples Betriebssystem iOS und dessen Browser Safari, als auch die oft eher schlechte Umsetzung einer WebApp, durch fehlender Funktion der Navigationsbuttons des Browsers oder der URL-Leiste.

Anhand einer Beispiel WebApp, einer simplen Online-Shop Anwendung, wird der Entwicklungsprozess von einer "normalen" Website zu einer progressiven WebApp dargelegt, sowie deren Verwendung von modernen Webtechnologien.

Vorallem spielen hier Googles vorrangetriebene Entwicklung von sogenannten Service-Workern eine große Rolle, welche die wichtigsten Funktionen und Merkmale einer progressiven WebApp möglich machen.

Inhaltsverzeichnis



3 Geschichte und heutige Technologien

Bevor auf die Geschichte und verschieden Technologien eingegangen werden kann, muss zwischen zwei Typen von Sprachen unterschieden werden: Clientseitige Sprachen, welche beim Benutzer im Browser ausgeführt werden, und serverseitige Sprachen, welche der Server ausführt, bevor oder während einer Anfrage. Außerdem gibt es weitere Standards, die zum Beispiel der Kommunikation zwischen Client und Server dienen.

3.1 Clientseitige Sprachen

HTML. Hypertext Markup Language. Von Beginn des "Internets" an ist HTML die grundlegende Sprache zum Struckturieren einer Webseite. Seit der Urversion von 1992 hat sich HTML stark verändert und weiterentwickelt. Damals war Hauptbestandteil nur der Text und dessen Verlinkung, auch bekannt als Hypertext-Referenzierung.[16]

Über die Jahre hinweg hat W3C HTML zu HTML5 respektive HTML5.1(seit Ende 2016 [9]) weiterentwickelt. Im Vordergrund stand hierbei vor Allem die strikte Trennung von Struktur und Design. In früheren Versionen war es zum Beispiel möglich mit `<center>` einen Text zu zentrieren. Dies soll, genauso wie alle anderen designspezifischen Tags und Attribute mittels CSS gelöst werden.

In der aktuellsten Revision, HTML5.1, wurde unter Anderem die Einbindung von "responsive Images", Bildern, welche dem Browser in verschieden Auflösungen zur Verfügung gestellt werden, sodass der Browser nur die passende Auflösung laden muss, möglich gemacht. Es wird sich demnach auf die Unterstützung der große Vielfalt an Endgeräten und somit verschiedenen Bildschirmauflösungen und -verhältnissen konzentriert, welches besonders bei WebApps benötigt wird, da diese sowohl auf mobilen Endgeräte, als auch mit herkömmlichen Rechnern benutzbar sein sollen. [2]

JavaScript, kurz auch JS. Ursprünglich LiveScript genannt wurde JavaScript entwickelt um dynamisches HTML zu erlauben. Hierbei wird das HTML-Dokument nach dem Laden beim Benutzer verändert. Der Name JavaScript rührt daher, dass LiveScript in Kooperation von Netscape und Sun Mircosystems entwickelt wurden. Um die Bekanntheit von Sun's Sprache Java zu nutzen wurde LiveScript in JavaScript umbenannt, obwohl es eigentlich wenig mit Java sowohl vom Syntax her, als auch vom Einsatzgebiet und Entwicklung. [17]

Heutzutage ist JavaScript Kernbestandteil von Webapps und ist für jegliche clientseitige Aktivität zuständig.

Neben JS existieren bzw. existierten auch weitere Skriptsprachen, wie Silverlight, Flash und JavaApplets. Alle sind mittlerweile als obsolet markiert und stellen durch Sicherheitslücken ein hohes Sicherheitsrisiko dar.[13] Chrome hat diese, wie andere Browser auch, bereits standardmäßig deaktiviert oder sogar ganz entfernt.[6]

Leider wird im Volksmund JavaScript häufig mit eben genannten JavaApplets verbunden, da die Sprache Java einen eher größeren Bekanntheitsgrad hat als Java. Daher rühren leider auch Gerüchte über große Sicherheitslücken durch JavaScript, obwohl eigentlich die JavaApplets die Sicherheitslücken beinhalten.

Da jeder Browser eine etwas andere Implementierung von JS verfolgt hat war es anfangs kompliziert alle Browser gleichzeitig zu unterstützen. Die European Computer Manufacturers Association, kurz ECMA, hat seit 1997 versucht JavaScript in einer Spezifikation zu vereinheitlichen und eben diese Probleme zu beseitigen. Mit Erfolg, denn seit der Version 5.1 aus dem Jahr 2011 ist JS, beziehungsweise ECMAScript eine einheitliche Sprache. ECMAScript2015, die sechste Version der Spezifikation, ist in den meisten beliebten Browsern bereits vollständig oder teilweise unterstützt und bildet die Grundlage für Service-Worker, welche in WebApps zwingend benötigt werden. Genauso beinhaltet diese Version auch die Möglichkeiten Klassen zu definieren und objektorientiert in JS zu entwickeln.[15]

JSON, JavaScript Object Notation, ist ein Standard um Objekte zwischen verschiedenen Programmiersprachen zu serialisieren. Dies wird unter anderem dafür genutzt um zwischen zwei Kommunikationspartnern, welche jegliche Programmiersprache für deren Software benutzen können, zu kommunizieren.

Vorteile von JSON gegenüber anderen Spezifikationen für einfachen Datenaustausch sind zum einen, dass es sowohl für Mensch, als auch für Maschine einfach zu lesen und zu interpretieren ist, zum anderen in praktisch jeder Programmiersprache ein Parser existiert um JSON-Objekte in respektive Objekte umzuwandeln.[7]

Andere beliebte oder zuvor beliebte Standards zum Austausch von Daten sind zum Beispiel YAML und XML. Letzteres wird jedoch immer häufiger durch JSON ersetzt, da XML in der Interpretierung weit aus aufwendiger ist und insgesamt für die gleichen Informationen mehr Rohdaten zur Darstellung benötigt.

AJAX, Asynchronous JavaScript and XML, ist ein, mittlerweile, Kernbestandteil von JS. Diese Funktion macht es möglich asynchron weitere Anfragen an den Server zu schicken, dessen Inhalt kann zum Beispiel das Aktualisieren von bereits angezeigten DAten sein, oder auch Benutzereingaben sein, um das Abschicken von Formularen ohne ein Neuladen der gesamten Seite möglich zu machen. Ursprünglich wurde für die Kommunikation XML benutzt, da aber aus im Abschnitt zuvor genannten Gründen es praktischer ist JSON zu verwenden, wird normalerweise nur noch JSON hierfür eingesetzt.[14]

WebSockets, werden, im Gegensatz zu AJAX-Verbindungen, aufrecht erhalten, um so schnell wie möglich Daten zwischen dem Server und dem Client austauschen zu können. Diese Technologie eignet sich vor allem für Echtzeitanwendungen, wie Livechats und Browserspiele, sind aber für WebApps nicht zwingend notwenidig.[25]

Responsive web design ist ein Begriff dafür, dass vorallem durch die große Vielfalt an Displaygrößen und -auflösungen geforderte Möglichkeit, eine Webseite zu haben, welche für alle Auflösungen passt. Dank CSS3 ist es möglich für verschiedene Displaygrößen verschiedene Designregeln festzulegen. Somit kann sich die Webseite immer an die benötigte Umgebung anpassen.

Josh Clark hat dieses Konzept in einem Satz ziemlich gut zusammengefasst: “Content is like water”[21]. Wenn man Wasser in einen Becher füllt, hat das Wasser die Form des Bechers, wenn man das was wiederum in eine Flasche, hat es die Form der Flasche. Genau so sollte laut ihm eine gute responsive Webseite sein.

3.2 Serverseitige Sprachen

PHP, "PHP: Hypertext Preprocessor", mittlerweile, mit über 80% der Websites, die am weitesten verbreitetste serverseitige Skriptsprache.[11]

PHP1, damals noch für *Personal Home Page Tools*, wurde als Ersatz zu Perlskripten von Rasmus Lerdorf entwickelt. Mit PHP3, welches von Andi Gutmans und Zeev Suraski entwickelt wurde(Lerdorf wurde als Entwickler auch eingestellt), wurde die Sprache von Grund auf neu entwickelt und unter dem rekursiven Akronym *PHP: Hypertext Preprocessor* veröffentlicht.

Mit PHP4 war es zudem möglich objektorientiert zu entwickeln, welches mit PHP5 weiter verbessert wurde.

Die aktuellste Version ist PHP7.1. Mit PHP7 kamen, 11 Jahre nach PHP5, vorallem eine erheblich verbesserte Performance und sowie einige neue Features, wie die Spezifizierung von Datentypen, dazu.[19]

4 Die (progressive) WebApp

4.1 Was ist eine (progressive) Webapp?

Eine WebApp ist tatsächlich schon länger im Internet zu finden als man bei dem sehr modern anmutenden Begriff "App", vermuten würde. Tatsächlich ist eine WebApp im Groben eine Internetseite, welche sich wie ein Programm verhält. Demnach sind alle Internetseiten, welche nicht komplett statisch sind, WebApps. Als eines der ersten Beispiele zählt "SPIRES-HEP", einem Webinterface zum Zugriff auf eine Datenbank der Stanford Universität im Jahr 1991. Damals wurde ein Vorläufer des heutigen HTTP-GET Verfahrens benutzt, um Daten der Benutzer an den Server zu schicken.

Trotzdem war dieses Interface auf Seiten des Clients noch statisch, denn nachdem einmal die Seite geladen wurde, konnte an dieser nichts mehr verändert werden. Auf der Seite des Servers hingegen war diese Seite dynamisch, denn je nach Anfrage des Benutzers hat der Server einen anderen HTML-Code generiert.

Eine der ersten von der breiten Öffentlichkeit wahrgenommenen WebApp ist die Suchmaschiene Yahoo! gewesen, welche ebenfalls von 2 Studenten der Stanford Universität entwickelt wurde und ursprünglich als Verzeichnis von ihren persönlichen Lesezeichen.[24]

Seitdem der Begriff AJAX in JavaScript Einzug erhalten hat, ist es möglich Daten vom Webserver abzufragen, ohne dabei die komplette Seite neu laden zu müssen. Seit 2005 fingen große Unternehmen, wie Google und ... an ihre Dienste immer interaktiver zu gestalten. Nebst JavaScript sind einige andere Technologien von verschieden Firmen ausprobiert worden, wie Flash von Adobe und Silverlight von Microsoft. Letztendlich konnte sich aber doch JavaScript, beziehungsweise ECMAScript durchsetzten.[23]

Die eigentliche Natur hinter dem Namen "Web Anwendung" sieht man an den Online-Office Varianten, sowohl von Microsoft, als auch von Google. Hier wurden Programme, welche von einem herkömmlichen Rechner bekannt waren in das Internet transferiert und um weiteren Funktionen, wie etwa der Möglichkeit mit mehreren Personen an einer Datei gleichzeitig arbeiten zu können, ergänzt.

Der Begriff “WebApp” wird mittlerweile von einigen Browsern, wie Google Chrome und Mozilla Firefox auch für eine spezielle Art deren Browser-Erweiterungen, beziehungsweise Plugins verwendet. Diese können, wie ein normales Programm in einem komplett selbst zur Verfügung stehenden Fenster geöffnet werden und sich wie ein natives Programm verhalten.[1]

Selbiges versucht Google auch auf mobilen Endgeräten durchzusetzten, mit Hilfe von “progressiven WebApps”, diese sind ähnlich zu den zuvor genannten WebApps, zur Zeit nur unter Android, Internetseiten, welche aussehen, als seien sie native Apps. Können aber auch mit Funktionen wie etwa einem Offline-Modus oder Push-Benachrichtigungen aufwarten. Jedoch begrenzt Google den Standard nicht nur auf mobile Endgeräte, sondern lässt die Kernfunktionen wie Service-Worker auch in Browsern auf herkömmlichen Rechnern zu, und macht somit ähnliche Funktionalität auch auf Internetseiten möglich, welche auf den ersten Blick nicht nach einer WebApp aussehen.[20]

4.2 Vor- und Nachteile

Progressive WebApps sind zum Zeitpunkt der Recherche leider noch kein fertiger Standard. Dank Googles starken Ethusiasmus die Entwicklung fortzutreiben sind progressive WebApps in der jetztigen Form bereits, sowohl am herkömmlichen Rechner, als auch an mobilen Endgeräten mit Google Chrome und Windows- oder Android-Betriebssystem völlständig unterstützt. Auch Mozilla Firefox unterstützt seit Version 44 nicht nur Service-Worker und somit die Grundlage hinter progressiven WebApps, sondern auch die meisten anderen Funktionen die für WebApps benötigt werden.[3] Auch in Microsoft neuem Browser EDGE sind Service-Worker und andere Funktionen für WebApps bereits in aktiver Entwicklung.

Das Team von Webkit, der Webplatform auf der unter anderem Safari basiert, jedoch hat diese erst in ihren 5-Jahres-Plan aufgenommen. Zuerst etwas zurückhaltend mit einem inoffiziellem Kommentar “Service Worker: People think they want it, some of them actually do want it. We should probably do it.”[5], später dann zu “Service Worker: Becoming a more frequent request. We should do it.”[12] geändert. Somit werden progressive WebApps vorerst ein Privileg sowohl für Androidbenutzer mit Chrome als Browser sein, als auch Desktopbenutzer mit Google Chrome oder Mozilla Firefox. [8]

Um aus einer bestehenden Webseite eine progressive WebApp zu machen ist in einigen wenigen Schritten möglich. Zum einen muss die Seite komplett über HTTPS ausgeliefert werden, was in der heutigen Zeit, auch dank Initiativen wie Let’s Encrypt, schon sehr weit verbreitet ist, zum anderen müssen zwei kleine Dateien erstellt werden, ein Service-Worker für Offline-Funktionalität und ein Manifest mit Icons in verschiedenen Auflösungen. Durch das bloße Hinzufügen dieser Dateien, muss an keiner anderen Stelle der Webseite etwas geändert werden, was die volle Kompatibilität mit Browser ohne die oben genannten Funktionen aufrecht erhält, denn diese Dateien werden von diesen Browsern einfach ignoriert.

Ein anderes Argument ist die oft schlechte Umsetzung einer WebApp. Zum einen sind diese oft Single-Page-Anwendungen und nicht jeder Entwickler implementiert die Funktion der Vor- und Zurück-Buttons des Browsers korrekt, so, dass diese innerhalb der Single-Page-Anwendung agieren und nicht auf die vorherige Seite zurückführen. Zum anderen ist es meist nicht mehr möglich eine URL zu

einer bestimmten Ansicht zu bekommen, denn auch hier sorgt die Single-Page-Anwendung dafür, dass nur eine URL für die gesamte Anwendung verfügbar ist.

Es gibt jedoch einige WebApps, bei denen diese Punkte gut umgesetzt wurden, so ändert sich zum Beispiel die Addressleiste passend zum aktuellen Kontext und beinhaltet eine URL welche beim direkten Aufruf auf diese Seite führt. Ein Beispiel hierfür wäre materialUP[1]. Leider ist die Entwicklung einer solchen WebApp wieder mit mehr Aufwand verbunden, wobei dieser weitaus niedriger bleibt, als bei nativen Apps, welche zusätzlich für mehrere verschiedene Platformen entwickelt werden müssten.

Je nach Einsatzgebiet der WebApp, kann es gewollt sein, dass sich beispielsweise die URL-Leiste nicht ändert.

5 Entwicklung einer progressiven WebApp

5.1 Vorbereitung

Zur Vorbereitung gehören neben der Einrichtung der Entwicklungsumgebung und des Webservers auch die Sammlung benötigter Bibliotheken.

Bei der Einrichtung des Webservers wurde ein Ordner für dieses Projekt samt passender Zugriffsrechte und gültigen SSL-Zertifikat erstellt. Dieses Zertifikat ist nicht nur für die Funktionalität von Service-Workern, bei welchen eine Verbindung via SSL Pflicht ist, sondern auch für den generellen Datenschutz des Nutzers wichtig.

Als einfachste Serversoftware ist hier Caddyserver[2] von Vorteil. Dieser bietet nicht nur einfach Erweiterbarkeit, wie zum Beispiel für PHP, und gute Performance, sondern unterstützt schon das neue Protokoll HTTP/2 und kann automatisch SSL-Zertifikate für die eingerichteten Domains austellen lassen.
Da eine Entwicklung ohne jegliche Bibliotheken viel zu lang dauern würde und nicht praktikabel ist wurden auch hier einige Bibliotheken verwendet:

Materialize CSS[3]: Dies ist eine CSS/JS-Bibliothek, welche jegliche Designelemente Googles Material Design Definition zur Verfügung stellt. Somit lassen sich ohne viel Aufwand, mittels HTML und passenden CSS-Klassen responsive Webseiten erstellen.

jQuery[4] ist eine der beliebtesten JavaScript-Bibliothek[10], welche viele häufig benötigten JavaScript-Funktionen in einfachere, schnell zu benutztende Funktionen vereint.

Dexie.js[5]: Um auf eine der beliebtesten Web Datenbanken, IndexedDB zuzugreifen bietet dexie.js einfache, objektorientierte Methoden und Klassen.

Einige kleinere JavaScript Funktionen wurden zusätzlich verwendet um zum Beispiel Hashfunktionen wie md5 auszuführen.

[1] `Material Up: https://material.uplabs.com/`
[2] `Caddyserver` - The HTTP/2 web server with automatic HTTPS: `https://caddyserver.com/`
[3] `Materialize Frontend-Framework: http://materializecss.com/`
[4] `jQuery JavaScript Library: https://jquery.com/`
[5] `dexie.js: http://dexie.org/`

5.2 Struktur

5.2.1 JavaScript

In JavaScript wird nach Möglichkeit das “Model View Controller”[18] Designpattern angewendet. Es soll hiermit der, Teil der die eigentlichen Daten verwaltet, von dem Teil, der den View, beziehungsweise die Darstellung, kontrolliert, trennen. Jediglich der Controller “kennt” beide Teile und sorgt dafür, dass diese zusammenarbeiten.

5.2.2 Datenbank

In der gesamten WebApp gibt es zwei verschiedene Datenbanken, einmal die IndexedDB, welche clientseitig agiert, nur die Daten verwaltet die wirklich relevant für den Benutzer sind und vor allem für die Gewährleistung der Offline-Funktionalität verantwortlich ist. Diese Datenbank hat eine ähnliche Struktur zu den Klassen in JavaScript.

Zum anderen eine MySQL-Datenbank, welche auf dem Server läuft und die gesamten Daten von allen Benutzern beinhaltet, beziehungweise die verwaltet.

5.2.3 Klassen in PHP

In PHP werden die Klassen sehr stark an die Tabellen aus der MySQL-Datenbank angelehnt, denn die Klassen sollen einzelne Datensätze dieser Tabellen repräsentieren können. Jeglich Form von Darstellung soll clientseitig passiern, daher kann ich PHP keines der Designpatterns für objekt orientierte Programmierung angewandt werden, weil PHP nur Daten in “Rohformat” oder verarbeitetem Rohformat zurückgibt.

5.2.4 API

–¿ API Diagramm? Die API ist die Grundlage zur Kommunikation zwischen Server und Client. Sie besteht aus einzelnen Ordnern, welche wie die PHP-Klassen strukturiert sind. Innerhalb dieser Ordner gibt es für jede Funktion, welche durch die API auf der Klasse ausgeführt werden soll, ein PHP-Skript, welches genau diese Funktion ausführt und zuvor auf Eingabefehler prüft.

5.3 Entwicklungsprozess

Die Reihenfolge der einzelnen Schritte sind in den meisten Fällen persönliche Präferenz. Oft ist bereits eine Webseite vorhanden, falls diese, wie in diesem Beispiel, bereits eine responsive Webseite ist, demnach sowohl auf normalen Rechnern, als auch auf mobilen Endgeräten gut zu Bedienen ist, kann diese beibehalten werden. Man sollte jedoch möglichst versuchen eine Single-Page-Webanwendung[22] aus der Webseite zu bauen, um die Illusion einer nativen Anwendung zu gewährleisten.

5.3.1 Client Grundlagen

Ursprünglich war sowohl die Menüführung, als auch der Inhalt in einer HTML-Datei. Um auf dieser einen Seite mehrere Seiten “emulieren” zu können wurde der

Bereich des Inhalts mehrfach hinzugefügt und versteckt. Es wird dann immer nur der Teil des Inhalts sichtbar gemacht, der die aktuelle Seite darstellt. Wenn man von einer normalen Webseite ausgeht, wurden hier mehrere HTML-Dateien in eine vereint. Bei kleineren Seiten führt das nicht merklich zu einem Nachteil, bei größeren kann es zu Performanceeinbußen kommen, da immer der gesamte Inhalt im Hintergrund geparst werden muss.

Um den entgegenzuwirken wird der passende HTML-Code erst während des Aufrufs der emulierten Unterseite erzeugt. Hierfür bietet die JS-Bibliothek `handlebars.js`[6] sogenannte Templates, dies sind Blaupausen für HTML-Code, welche mit Daten ausgefüllt werden können.

Zunächst wird die App-Shell, welche das immer gleichbleibende Grundgerüst der Anwendung, wie der Kopfzeile und der Navigationsleiste, darstellt, von der restlichen Anwendung getrennt. Somit erhalten wir ein "leeres" Grundgerüst, in dem ein `div` existiert, welches später mittels DOM-Manipulation mit, von zuvor genannten Blaupausen generierten, Inhalt gefüllt werden kann. Dies bildet die Datei `appUser.html`

5.3.2 Das Manifest

Das Manifest ist der Bestandteil einer progressiven WebApp, der es (unter anderem) möglich macht, diese zum Homescreen hinzuzufügen.

Es besteht aus einer JSON-Datei, welche in jedem HTML-Dokument einer WebApp im `<head>`-Bereich verlinkt sein muss. In dieser Datei werden neben dem Namen, Icons und der Start-URL auch Parameter zur Darstellung wie der Hauptfarbe und der Orientation festgelegt. Der Parameter `"display": "standalone"` macht aus einer normalen Webseite eine "echte" WebApp, indem es jegliche UI-Elemente des Browsers verbergt und die Webseite wie eine native App aussehen lässt.

5.3.3 Der Service-Worker

Grundlegend ist ein Service-Worker ein JavaScript-Skript, welches vom Browser "installiert" wird. Das heißt der Browser speichert das Skript und führt dieses, nicht nur solange eine Seite bzw. Tab offen ist, sondern auch darüber hinaus, aus.

In Google Chrome zum Beispiel funktionieren Service-Worker, genauso wie einige andere Funktionen, darunter Zugriff auf die Webcam oder den Standort, nur wenn die aufgerufene Seite ein "secure Origin" ist, hierbei *muss* die Seite unter anderem vollständig über SSL ausgeliefert werden. Ausnahmen hierfür sind zu Entwicklungszwecken `localhost` und `127.0.0.1`. Außerdem darf pro Webseite nur ein Service-Worker existieren.

Eine Aufgabe, die dieser Service-Worker erledigt, ist das sogenannte "Caching" von Dateien. Hierbei werden alle zur Offline Benutzung benötigten Resourcen in einem "Cache" zwischengespeichert. Somit kann die WebApp größtenteils auch offline verwendet werden.

Da die Entwicklung dieses Teils des Service-Workers oft mit viel reproduktiver Arbeit verbunden ist, haben die Google Chrome Entwickler eine Bibliothek[7] zur

[6] `handlebars.js`: `http://handlebarsjs.com/`

[7] `sw-precache` - GitHub: `https://github.com/GoogleChrome/sw-precache`

Verfügung gestellt. Mit dieser lässt sich nach eingestellten Regeln ein Service-Worker erstellen, der alle benötigten Resourcen “cachet”. Somit muss auf dem Entwicklungsrechner lediglich ein lokales Skript ausgeführt werden, welches alle zu cachenden Dateien im Service-Worker verlinkt.

Eine weitere Aufgabe des Service-Workers besteht darin, Push-Benachrichtigungen zu verwalten. Hierbei muss der Benutzer erst bestätigen, dass dieser Nachrichten erhalten darf. Durch eine Bestätigung hat dieser den Push-Dienst abonniert, und erhält von diesem Zeitpunkt an Push-Benachrichtigungen. Diese Bestätigung kann jederzeit wieder zurückgenommen werden, falls die Benachrichtigungen durch den Nutzer als störend empfunden werden. Hierauf wird im späteren Verlauf der Entwicklung weiter eingegangen.

5.3.4 View-Klassen

Durch ECMA-Script 2015 und 2016, kurz ECMA-Script 6 oder ES6, ist es möglich “echte” Klassen in JS zu schreiben. Diese werden ähnlich zu anderen Programmiersprachen wie folgt definiert:

```
class Foo {
    constructor(bar) {
        this.bar = bar;
    }

    foobar() {
        return 42;
    }
}
```

Als ersten Teil wird sich nun die “Account Einstellungen”-Seite vorgenommen, hier kann später neben Änderungen am Passwort und der Email-Adresse und anderen Optionen auch die Erlaubnis zum Senden bzw. Empfangen von Push-Benachrichtigungen gegeben, oder auch zurückgezogen werden.

Um eine gute Struktur im JS zu behalten, wird sich an den zuvor erklärten ES6-Klassen bedient. Für jeden “View”, also jede emulierte Unterseite, wird eine eigene “View-Klasse” erstellt. Diese beinhaltet, im Gegensatz zu Standart-Klassen, keine Daten, sondern beherbergt Methoden und Eigenschaften, welche zur Darstellung dieser Unterseite nötig sind.

Im Konstruktur werden alle Daten gesammelt die schon vor dem Ausführen der Unterseite benötigt werden. Hierzu gehört unter anderem das Kompilieren der Handlebars.js-Blaupause.

Neben einer Funktion `showView()`, welche die Unterseite im Inhaltsbereich des Grundgerüsts einfügt und somit anzeigt, beinhaltet diese Klasse auch Funktionen welche bei Interaktion mit der Seite ausgeführt werden. Um diese “Befehle” an das Modell weiterzugeben wurden zuvor sogenannte Callback-Funktionen durch den Kontroller übergeben, diese sind Funktionen des Modells, die die View-Klasse zwar nicht kennt, sie jedoch aufrufen kann, um das Modell über beispielsweise einen Klick auf einen Button zu informieren.

Genauso wie hier kann mit den weiteren Unterseiten verfahren werden. Wichtig ist jedoch, dass innerhalb der View-Klassen keine AJAX-Anfragen zum Server geschickt werden sollen, um Daten abzufragen. Dieser Teil soll von anderen Klassen später geregelt werden.

5.3.5 Datenmodell-Klassen

Um nun auch mit Daten arbeiten zu können, wird in diesem Schritt das Datenmodell erstellt. Diese Klassen werden zum Beispiel durch den Kontroller benutzt, um die, vom Server erhaltenen Daten, in Objekte umzuwandeln und somit einfacher verwenden kann.

Im Beispiel des Online-Shops umfasst das Modell die Klassen `FilamentType`, `Order`, `User`. Solche Klassen wurden in der Basis-Version nicht verwendet, in dieser wurde der empfangene JSON-Code direkt mittels einer HTML-Blaupause auf der Seite angezeigt.

Ein Vorteil diese Klassen ist es, dass diese auch ohne eine Anbindung an einen Server funktionieren können. Da die serverseitige Entwicklung erst im Anschluss folgt, werden die Daten nicht vom Server empfangen, sondern in einer IndexedDB gespeichert. Später dient diese Datenbank als ein Zwischenspeicher für die empfangenen Daten des Servers um so die Offline-Funktionalität zu gewährleisten.

Die Implementierung einer solchen Datenbank setzt vorraus, asynchrones JavaScript zu programmieren, welches das gesamte JS mit einigen Callback-Funktionen mehr verkompliziert.

5.3.6 Der Controller

Da nun sowohl das Model als auch der View Klassen haben, kann der Controller entwickelt werden. Dieses JS-Skript initiiziert das Laden des Models aus der IndexedDB und gibt diese Daten an die passende View-Klasse weiter, sobald diese benötigt werden, das heißt, sobald der Benutzer die emulierte Unterseite aufruft.

Desweitern gibt der Kontroller die Interaktionen des Benutzers mit dem View and das Model weiter um die IndexedDB nach den Änderungen des Benutzers zu aktualisieren.

5.3.7 Serverseitige Grundlagen

Als erste Grundlage einer jeden Webseite steht die Datenbank. Diese ist in diesem Beispiel bereits vorhanden und auch schon mit Daten gefüllt, so, dass man eine Kopie dieser Datenbank gut zum Entwickeln und Testen benutzen kann. Die Klassen in PHP halten sich, wie oben bereits genannt sehr stark an das verwendete Datenbankschema. Demnach sind diese Klassen Repräsentanten in PHP für die einzelnen Datensätze in der Datenbank. Trotzdem beherbergen sie einige weiter Funktionen, um zum Beispiel Daten des Benutzers zu Verarbeiten und zu Validieren.

5.3.8 API

Eine richtige API ist bisher noch nicht implementiert worden, es wurde zwar mit dem Server schon via JSON kommuniziert, aber dies ging von einem PHP-Skript aus, welches nicht gerade strukturiert war.

Die hier verwendete API ist ähnlich zu einer "RESTful-API"[4], jedoch ist es nicht immer möglich diese auf jedem beliebigen Webserver zu implementieren, da PHP hierbei Zugriff auf verschiedene HTTP-Header, wie `GET`, `PUT`, `DELETE`, `INSERT`, `POST`, `usw.` benötigt. Anstatt HTTP-Header auf Ordner anzuwenden, werden in diesem Beispiel für jede Funktion die auf einen dieser API Ordner, beispielsweise `/api/users/`, angewendet werden soll, wird ein PHP-Skript erstellt, dass den passenden Namen trägt: `create.php`.

Diesem Skript kann nun mittels Standard-Headern die zu verwendenden Daten übergeben werden. Die Antwort des Server erfolgt hierbei stets mittels JSON, sodass der JavaScript-Client ziemlich schnell die Antwort interpretieren und weiterverwenden kann.

5.3.9 Vollständige Offline-Fähigkeit

Nachdem durch den Einbau eines Service-Workers bereits alle statischen Inhalte und Ressourcen offline verfügbar gemacht wurden, sollen auch die dynamischen Ressourcen offline einsehbar gemacht werden. Hierfür wurde zuvor die IndexedDB implementiert. Sobald der Benutzer nun versucht Daten anzeigen zu lassen versucht die passende Model-Klasse zunächst neuste Daten vom Server abzufragen, falls dies scheitern sollte, aber bereits ein älterer Stand in der IndexedDB existiert benutzt die Klasse diesen, informiert jedoch den Controller darüber, dass es sich um eine ältere Version der Daten handelt, damit die View-Klassen den Benutzer darüber in Kenntnis setzten können.

Damit jedoch in der IndexedDB im Offline-Modus eine älterer Stand der Daten sein kann, werden bei jeder Anfrage an der Server das Ergebnis in dieser Datenbank strukturiert gespeichert, samt dem Zeitpunkt, von wann diese Daten stammen.

5.3.10 Push-Benachrichtigungen

Um den Benutzter des Online-Shops angemessen über den Stand seiner Bestellungen informieren zu können sollen Push-Benachrichtigungen in der progressiven WebApp Einzug erhalten.

Da diese Benachrichtigungen auf einem vertraulichen Level sind, müssen diese verschlüsselt werden. Deshalb benötigt sowohl jeder Client, als auch der Server jeweils ein Schlüsselpaar. Für den Server passiert die Generierung dieses RSA-Schlüsselpaars nach Möglichkeit nur einmal, denn dessen öffentlicher Schlüssel wird fest in den JavaScript eingebunden.

Wenn nun ein Benutzer in den Einstellungen die Push-Benachrichtigungen aktivieren möchte, fragt der Browser diesen, ob er der Webseite, beziehungsweise WebApp, zulassen möchte. Ist diese Anfrage bestätigt worden, generiert der Client, genauso wie der Server auch, ein RSA-Schlüsselpaar und sendet den öffentlichen Schlüssel, samt sogenannter Entpunkt Informationen an den Server. Je nach benutzten Browser werden Push-Benachrichtigungs Server zur Verfügung gestellt,

welche die Nachrichten jederzeit, verschlüsselt, empfangen und versuchen an den Empfänger weiterzuleiten.

Sollte dies nicht funktionieren, hält der Server diese Nachricht solange zurück, bis eine zuvor definierte Zeit, beispielsweise 2 Tage, abgelaufen ist, oder der Client wieder erreichbar ist.

Der eigene Server kennt, somit die Endpunkte der einzelnen Benutzer und kann diesen mit Hilfe der Push-Severn verschlüsselt Nachrichten schicken. Diese müssen noch nicht einmal speziell für Push-Benachrichtigungen gedacht sein, sondern können den Client auch über andere wichtige Dinge informieren.

Im Online-Shop Beispiel wird der JSON-Kodierte Code, der als Push-Nachricht beim Client angekommen ist, zuerst interpretiert und daraufhin die passende Benachrichtigung angezeigt, mit welcher der Benutzer genauso Interagieren kann, wie mit einer nativen Benachrichtigung. So kann dieser die Benachrichtigung anklicken und wird automatisch zur passenden Bestellung weitergeleitet.

6 Fazit

Trotz der teils noch mangelhaften Unterstützung der Technologien, welche notwendig sind um WebApps oder progressive WebApps möglich zu machen. Vor allem Webkit ist hier mit der Entwicklung leider erst ziemlich am Anfang, was einen der größten Vorteile einer WebApp

7 Anhang

7.1 Literaturverzeichnis

[1] Chromium Google. *Chrome Web Store*. Englisch. 23. Apr. 2017. URL: https://chrome.google.com/webstore/category/apps (besucht am 23. 04. 2017).

[2] Philippe le Hegaret. *HTML 5.1 IS THE GOLD STANDARD*. Englisch. 17. Nov. 2016. URL: https://www.w3.org/blog/2016/11/html-5-1-is-the-gold-standard/ (besucht am 03. 04. 2017).

[3] jakearchibald. *Is SERVICEWORKER ready?* Englisch. 6. Feb. 2017. URL: https://jakearchibald.github.io/isserviceworkerready/ (besucht am 23. 04. 2017).

[4] Henrik Joreteg. *Representational state transfer*. Englisch. 20. Apr. 2017. URL: https://en.wikipedia.org/wiki/Representational_state_transfer (besucht am 23. 04. 2017).

[5] Henrik Joreteg. *Somewhat patronizing mention of Service Worker in WebKit's 5 year plan*. Englisch. 6. Dez. 2015. URL: https://twitter.com/HenrikJoreteg/status/673759932714500096?ref_src=twsrc%5Etfw&ref_url=http%3A%2F%2Fdeveloper.telerik.com%2Ffeatured%2Fwhat-progressive-web-apps-mean-for-the-web%2F (besucht am 23. 04. 2017).

[6] Sascha Koesch. *Chrome 56 deaktiviert Flash*. 27. Jan. 2017. URL: http://de.engadget.com/2017/01/27/chrome-56-deaktiviert-flash/ (besucht am 03. 04. 2017).

[7] N/a. *Introducing JSON*. Englisch. URL: http://json.org/ (besucht am 03. 04. 2017).

[8] TJ VanToll. *What Progressive Web Apps Mean for the Web*. Englisch. 14. Dez. 2015. URL: http://developer.telerik.com/featured/what-progressive-web-apps-mean-for-the-web/ (besucht am 23. 04. 2017).

[9] W3C. *HTML 5.1*. Englisch. 1. Nov. 2016. URL: https://www.w3.org/TR/2016/REC-html51-20161101/ (besucht am 02. 04. 2017).

[10] W3Tech. *Usage of JavaScript libraries for websites*. Englisch. 5. Apr. 2017. URL: https://w3techs.com/technologies/overview/javascript_library/all (besucht am 05. 04. 2017).

[11] W3Tech. *Usage of server-side programming languages for websites*. Englisch. 3. Apr. 2017. URL: https://w3techs.com/technologies/overview/programming_language/all (besucht am 03. 04. 2017).

[12] webkit. *Is SERVICEWORKER ready?* Englisch. 22. März 2016. URL: https://trac.webkit.org/wiki/FiveYearPlanFall2015 (besucht am 23. 04. 2017).

[13] Markus Werner. *Weg mit Flash: Der Flash Player ist ein Sicherheitsrisiko*. 6. Apr. 2016. URL: https://www.basicthinking.de/blog/2016/04/06/flash-player-sicherheit/ (besucht am 03. 04. 2017).

[14] Wikipedia. *Ajax (Programmierung)*. 1. März 2017. URL: https://de.wikipedia.org/wiki/Ajax_(Programmierung) (besucht am 03. 04. 2017).

[15] Wikipedia. *ECMAScript*. Englisch. 18. Apr. 2017. URL: https://en.wikipedia.org/wiki/ECMAScript (besucht am 21. 04. 2017).

[16] Wikipedia. *Hypertext Markup Language*. 1. Apr. 2017. URL: https://de.wikipedia.org/wiki/Hypertext_Markup_Language (besucht am 02. 04. 2017).

[17] Wikipedia. *JavaScript*. 22. März 2017. URL: https://de.wikipedia.org/wiki/JavaScript (besucht am 03. 04. 2017).

[18] Wikipedia. *Model View Controller*. 20. Apr. 2017. URL: https://de.wikipedia.org/wiki/Model_View_Controller (besucht am 22. 04. 2017).

[19] Wikipedia. *PHP*. 20. März 2017. URL: https://de.wikipedia.org/wiki/PHP (besucht am 03. 04. 2017).

[20] Wikipedia. *Progressive Web App*. 2. März 2017. URL: https://de.wikipedia.org/wiki/Progressive_Web_App (besucht am 04. 04. 2017).

[21] Wikipedia. *Responsive web design*. Englisch. 4. Apr. 2017. URL: https://en.wikipedia.org/wiki/Responsive_web_design (besucht am 23. 04. 2017).

[22] Wikipedia. *Single-Page-Webanwendung*. 19. März 2017. URL: https://de.wikipedia.org/wiki/Single-Page-Webanwendung (besucht am 08. 04. 2017).

[23] Wikipedia. *Web application*. Englisch. 18. Apr. 2017. URL: https://en.wikipedia.org/wiki/Web_application (besucht am 23. 04. 2017).

[24] Wikipedia. *Webanwendung*. 8. März 2017. URL: https://de.wikipedia.org/wiki/Webanwendung (besucht am 04. 04. 2017).

[25] Wikipedia. *WebSocket*. 23. März 2017. URL: https://de.wikipedia.org/wiki/WebSocket (besucht am 03. 04. 2017).

[26] Wissenschaftliches-Arbeiten.org. *Erklärung über die eigenständige Erstellung der Hausarbeit*. URL: https://www.wissenschaftliches-arbeiten.org/hausarbeit/aufbau/die-erklaerung.html (besucht am 15. 04. 2017).

7.2 Quellcode

Zu finden auf der beigefügten CD-ROM oder online unter https://github.com/yannick9906/3d-print-shop.

Auf der CD-ROM befindet sich außerdem der LaTeX-Quellcode, mit welchem dieses Dokument, beziehungsweise diese PDF-Datei, erstellt wurde.

8 Erklärung über die selbstständige Anfertigung der Arbeit

Hiermit erkläre ich, dass ich die vorliegende Hausarbeit selbständig verfasst und keine anderen als die angegebenen Hilfsmittel benutzt habe. Die Stellen der Hausarbeit, die anderen Quellen im Wortlaut oder dem Sinn nach entnommen wurden, sind durch Angaben der Herkunft kenntlich gemacht. Dies gilt auch für Zeichnungen, Skizzen, bildliche Darstellungen sowie für Quellen aus dem Internet.[26]

Mainz, den 24. April 2017

Yannick Félix