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 sind 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 können auf jedem Betriebssystem benutzt werden. 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.

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.

Tut mir sehr leid, aber der "Benutzerprofildienst" ist heute leider nicht verfügbar. Selbst der "Gruppenrichtlinienclient" kann hier leider nichts mehr ausrichten... Versuche es in 42 Minuten erneut.

Inhaltsverzeichnis

1 Kurzfassung **1**

2 Inhaltsverzeichnis **2**

3 Geschichte und heutige Technologien **3**
3.1 Clientseitige Sprachen . . . 3
3.2 Serverseitige Sprachen . . . 4
3.3 Webapp . . . 5

4 Entwicklung einer progressiven WebApp **5**
4.1 Vorbereitung . . . 5
4.2 Struktur . . . 6
4.2.1 JavaScript . . . 6
4.2.2 Datenbank . . . 6
4.2.3 Klassen in PHP . . . 7
4.2.4 API . . . 7
4.3 Entwicklungsprozess . . . 7
4.3.1 Client Grundlagen . . . 7
4.3.2 Das Manifest . . . 7
4.3.3 Der Service-Worker . . . 8
4.3.4 View-Klassen . . . 8
4.3.5 Modell-Klassen . . . 9
4.3.6 Synchronisierung mit dem Server . . . 9

5 Nachteile einer (progressiven) WebApp **10**

6 Fazit **10**

7 Anhang **11**
7.1 Literaturverzeichnis . . . 11
7.2 Quellcode . . . 12

8 Erklärung über die selbstständige Anfertigung der Arbeit **13**

3 Geschichte und heutige Technologien

Bevor auf die Geschichte und verschieden Technologien eingegangen wird, 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.[11]

Über die Jahre hinweg hat W3C HTML zu HTML5 respektive HTML5.1(seit Ende 2016 [5]) 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. [1]

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 syntaxtechnisch zu tun hat. [12]

Heutzutage ist JavaScript Kernbestandteil von Webapps und handelt jegliche clientseitige Aktivität.

Neben JS existieren bzw. existierten auch weitere Skriptsprachen, wie Flash und JavaApplets. Beide sind mittlerweile als obsolet markiert und stellen durch Sicherheitslücken ein hohes Sicherheitsrisiko dar.[8] Chrome hat diese, wie andere Browser auch, bereits standardmäßig deaktiviert.[2]

Die European Computer Manufacturers Association, kurz ECMA, hat seit 1997 versucht JavaScript in einer Spezifikation zu vereinheitlichen und somit die Programmierung für verschiedene Browser zu erleichtern. Version 5.1 aus dem Jahr 2011 hat dies vollständig erreicht und ist auch heute noch die meistunterstützte

Version von JS, beziehungsweise ECMAScript. ECMAScript2015, die sechste Version der Spezifikation, ist in den meisten beliebten Browsern bereits unterstützt und bildet die Grundlage für Service-Worker, welche in WebApps zwingend benötigt werden.[10]

JSON, JavaScript Object Notation, ist ein Standard um Objekte zwischen verschiedenen Programmiersprachen zu serialisieren. (Streng genommen ist JSON gar keine Sprache, sondern eine Art Protokoll um zwischen 2 Endpunkten zu kommunizieren, da aber der Ursprung der Spezifikation in der Sprache JavaScript liegt und JSON tatsächlich valides JS ist kann man es zu clientseitigen Sprachen zählen)

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.[3]

Andere 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.[9]

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.[18]

3.2 Serverseitige Sprachen

PHP, "PHP: Hypertext Preprocessor", mittlerweile, mit über 80% der Websites, die am weitesten verbreitetste serverseitige Skriptsprache.[7]

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 Programmieren, welches mit PHP5 weiter verbessert wurde.

Die aktuellste Version ist PHP7.1. Mit PHP7 kamen, 11 Jahre nach PHP5, vorallem eine verbesserte Performance und sowie einige neue Features, wie die Spezifizierung von Datentypen, dazu.[14]

3.3 Webapp

Die Funktion einer interaktiven Webseite setzt die Möglichkeit vorraus, dass der Webserver Daten empfangen kann, die der Benutzer eingegeben hat. Bereits in der Ur-Version von HTML war es möglich Werte an den Webserver zu schicken, ähnlich dem heutigen HTTP-GET Verfahren. Eine der ersten Seiten war “SPIRES-HEP”, ein Webinterface für eine Datenbank der Stanford Universität aus dem Jahr 1991.

Die erste von der Öffentlichkeit wahrgenommene Webanwendung war Yahoo, welches ebenfalls an der Stanford Universität von zwei Studenten entwickelt wurde.

Seit AJAX in JavaScript Einzug erhalten hat, ist es möglich auch dynamisch Teile der angezeigten Webseite zu verändern, oder sogar die Präsentationsschicht komplett auf den Client auszulagern, hierbei sendet während der normalen Benutzung der Server nur noch benötigte Informationen.[17]

Aktuell spricht man auch von “Progressiven Web Apps”, wenn diese nicht nur komplett clientseitig ausgeführt werden können, sondern auch Features wie ein Offline-Modus und Push-Benachrichtigungen enthalten. Außerdem können diese den Eindruck erwecken, eine native App zu sein, obwohl keine Installation benötigt wird. Leider werden diese bisher nur von Chrome und Firefox unterstützt, andere Browser haben zum Zeitpunkt der Recherche angekündigt progressive WebApps in Zukunft zu unterstützen.[15]

4 Entwicklung einer progressiven WebApp

4.1 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[1] 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[2]: 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.

[1] `Caddyserver` - The HTTP/2 web server with automatic HTTPS: `https://caddyserver.com/`
[2] `Materialize Frontend-Framework`: `http://materializecss.com/`

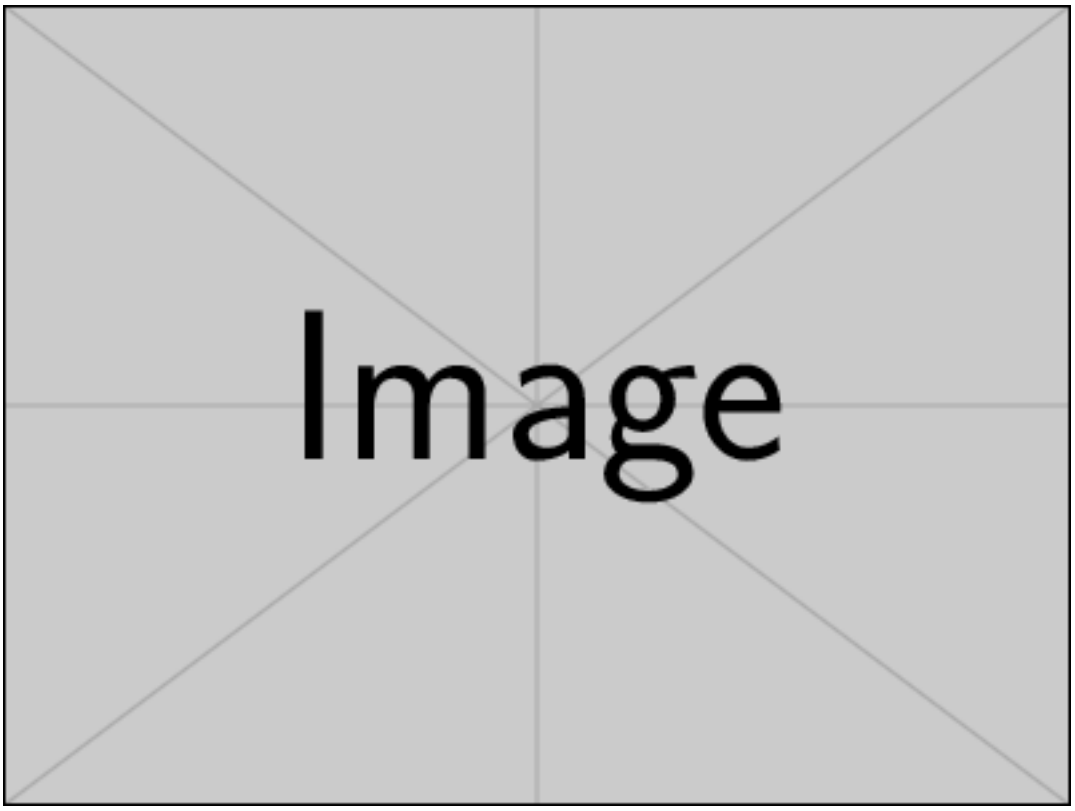

Abbildung 1: Klassendiagramm JavaScript

jQuery[3] ist eine der beliebtesten JavaScript-Bibliothek[6], welche viele häufig benötigten JavaScript-Funktionen in einfachere, schnell zu benutztende Funktionen vereint.

Dexie.js[4]**:** Um auf eine der beliebtesten Web Datenbanken, IndexedDB zuzugreifen bietet dexie.js einfache, objektorientierte Methoden und Klassen.

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

4.2 Struktur

4.2.1 JavaScript

In JavaScript wird nach Möglichkeit das “Model View Controller”[13] Muster angewendet. Es soll also 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.

4.2.2 Datenbank

–¿ Datenbankdiagramm

[3] jQuery JavaScript Library: https://jquery.com/
[4] dexie.js: http://dexie.org/

4.2.3 Klassen in PHP

–¿ Klassendiagramm

4.2.4 API

–¿ API Diagramm?

4.3 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[16] aus der Webseite zu bauen, um die Illusion einer nativen Anwendung zu gewährleisten.

4.3.1 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`[5] 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`

4.3.2 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

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

aus einer normalen Webseite eine “echte” WebApp, indem es jegliche UI-Elemente des Browsers verbergt und die Webseite wie eine native App aussehen lässt.

4.3.3 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[6] zur 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.

4.3.4 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;
    }
}
```

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

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.

4.3.5 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, dabei 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.

4.3.6 Synchronisierung mit dem Server

Um Offline-Funktionalität gewehrleisten zu können, werden alle Aktionen die der Benutzer in der WebApp ausführt zunächst in der lokalen Datenbank ausgeführt. Außerdem wird gespeichert, welche Aktion der Benutzer in welcher Reihenfolge getan hat. Diese Aktionen werden daraufhin, beziehungsweise sobald der Benutzer wieder mit dem Internet verbunden ist, an den Server gesendet um auf der

serverseitigen Datenbank ausgeführt zu werden. Der Server sendet nach erfolgreicher Ausführung die aktuellen Daten an den Client zurück. Dieser ersetzt die alten Daten in der lokalen Datenbank durch die neuen vom Server.

5 Nachteile einer (progressiven) WebApp

WebApps sind zum Zeitpunkt der Recherche leider noch kein fertiger Standard. Vor allem Google hat in letzter Zeit enorm die Entwicklung in Chrome und Chrome Android voran getrieben. In Firefox sind Bestandteile der WebApp, wie die Service-Worker unter aktiver Entwicklung; Das Team von Webkit, der Webplatform auf der unter anderem Safari basiert, hat diese in ihren 5-Jahres-Plan mitaufgenommen. Somit werden progressive WebApps vorerst ein Privileg für Androidbenutzer mit Chrome als Browser sein.[4]

Um aus einer bestehenden Webseite eine progressive WebApp zu machen genügen eigentlich schon die Verbindung über HTTPS, ein Service-Worker und ein Manifest. Die beiden letzteren Dateien können recht schnell hinzugefügt werden, ohne damit an anderen Stellen der Webseite für Probleme zu sorgen, denn Browser, die diese nicht unterstützen ignorieren diese Dateien einfach.

Ein anderen 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 richtig 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[7]

6 Fazit

Im Großen und Ganzen sind (progressive) WebApps eine zukunftsträchtige Technologie, dessen aktuelle Entwicklung auf jeden Fall Aufmerksamkeit zu wenden ist. Die aktuelle Unterstützung beschränkt sich zwar nur auf Chrome und Android, sollte aber im Laufe der nächsten Jahre zunehmen.

Es schadet auch jetzt nicht eine Webseite fit für eine WebApp zu machen, denn der Entwicklungsaufwand, im Gegensatz zu einer nativen App, ist um einiges niedriger.

[7] Material Up: https://material.uplabs.com/

7 Anhang

7.1 Literaturverzeichnis

[1] 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).

[2] 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).

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

[4] 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).

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

[6] 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).

[7] 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).

[8] 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).

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

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

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

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

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

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

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

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

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

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

[19] 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.[19]

Mainz, den 24. April 2017

Yannick Félix