hochschule mannheim

Konzipierung und Entwicklung einer Progressive Web App zum Herunterladen, Verwalten und Abspielen von Audio-Medien zur Offlinenutzung mit Angular

Martin Schalter

Bachelor-Thesis

zur Erlangung des akademischen Grades Bachelor of Science (B.Sc.)

Studiengang Informatik

Fakultät für Informatik

Hochschule Mannheim

19.08.2020

Betreuer

Prof. Dr. Thomas Specht, Hochschule Mannheim

Christian Perian, biblepool gUG

Schalter, Martin:
Konzipierung und Entwicklung einer Progressive Web App zum Herunterladen, Verwalten und Abspielen von Audio-Medien zur Offlinenutzung mit Angular / Martin Schalter. – Bachelor-Thesis, Mannheim: Hochschule Mannheim, 2020. 23 Seiten.

Schalter, Martin:
Developement of a progressive web app to download, manage and play audio files for offline use with Angular / Martin Schalter. –
Bachelor Thesis, Mannheim: University of Applied Sciences Mannheim, 2020. 23 pages.

Erklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Ich bin damit einverstanden, dass meine Arbeit veröffentlicht wird, d. h. dass die Arbeit elektronisch gespeichert, in andere Formate konvertiert, auf den Servern der Hochschule Mannheim öffentlich zugänglich gemacht und über das Internet verbreitet werden darf.

Mannheim, 19.08.2020

Martin Schalter

Abstract

Konzipierung und Entwicklung einer Progressive Web App zum Herunterladen, Verwalten und Abspielen von Audio-Medien zur Offlinenutzung mit Angular

—

Developement of a progressive web app to download, manage and play audio files for offline use with Angular

—

Inhaltsverzeichnis

Kapitel 1

Einleitung

In diesem Kapitel wird zuerst die Motivation und Zielsetzung der vorliegenden Thesis erläutert, damit der Leser sich einen ersten Eindruck verschaffen kann. Anschließend gibt es einen Überblick über alle nachfolgenden Kapitel.

1.1 Motivation

Das Webportal CROSSLOAD bietet viele christliche Medien zum Download an. Zur Nutzergruppe gehören Leute, die es gewohnt sind mit dem Computer oder Smartphone zu arbeiten, aber auch solche mit wenig Erfahrung im Umgang mit technischen Geräten. Das Portal wird sowohl von daheim im eigenen WLAN als auch unterwegs mit mobilen Daten genutzt.

Durch die mancherorts schlechte Mobilfunkabdeckung oder das beschränkte Datenvolumen möglicher Nutzer, könnte die Nutzung des Portals an vielen Stellen nicht möglich sein. Wenn man die Medien unterwegs abrufen möchte, müsste man sie sich vorher herunterladen und anschließend auf dem Gerät suchen. Für einige Nutzer ist das eine zu große Hürde und muss deswegen vereinfacht werden.

1.2 Zielsetzung

Ziel dieser Arbeit ist es für CROSSLOAD ein Konzept zur Offline Nutzung zu entwickeln und anschließend zu implementieren. Allen Nutzern soll eine kompfor-

table Nutzung des Webportals angeboten werden, sowohl auf Desktop-Computern, Tables oder Smartphones.

Dafür werden verschiedene Funktionen und Application Programming Interfaces (APIs) von Webbrowsern untersucht und verglichen. Die passendsten Funktionen werden ausgewählt und in das vorhandene Portal eingebunden. Dabei muss noch kein fertiges Produkt entstehen, aber die Machbarkeit gezeigt werden.

Schließlich stellt sich noch die Frage, ob die Funktionen und APIs der Webbrowser ausreichen, um alle Anforderungen zu erfüllen. Ist eine Progressive Web App (PWA) in der Lage für CROSSLOAD eine native App zu ersetzten?

1.3 Aufbau

Zuerst werden die Anforderungen an das Webportal im Blick auf die Offline Nutzung herausgearbeitet. Danach folgt die Konzeption zur Datenspeicherung und zum Herunteladen der Daten. Anschließend wird auf die Implementierung der ausgewählten Funktionen eingegangen. In Kapitel 6 folgt die Evaluation und kritische Beurteilung. Zuletzt werden noch einige Funktionen beleuchtet, die in Zukunft umgesetzt werden könnten, um eine noch bessere Nutzerfreundlichkeit zu bieten.

Kapitel 2

Grundlagen

Alle relevanten Grundlagen zum Verständnis dieser Thesis werden in diesem Kapitel erklärt. Dabei geht es um verwendete Begriffe, Technologien sowie Frameworks. Zuletzt wird CROSSLOAD, das zu erweiternde Webportal beschrieben und gezeigt.

2.1 Progressive Web App

Eine PWA ist eine Anwendung, die in modernen Webbrowsern läuft und dem Nutzer dabei eine Erfahrung ähnlich zu bekannten nativen Apps bietet (Sheppard 2017) (Rojas 2020). Die wichtigsten Funktionen dabei sind:

- Schnell: Die App startet schnell und bietet schnell die Möglichkeit zur Interkation (Hajian 2019) (Sheppard 2017)
- Zuverlässig: auf älteren Geräten funktionsfähig und auch ohne Internet nutzbar. Außerdem passt sich das Design an verschiedene Bildschirmgrößen an (responsive design) (Hajian 2019) (Sheppard 2017)
- Installierbar: nach dem Installieren wird ein Icon auf dem Homescreen angezeigt (Hajian 2019) (Sheppard 2017) (Rojas 2020)
- Benachrichtigungen: der Nutzer kann Benachrichtigungen bekommen, obwohl er die App nicht offen hat und aktiv benutzt (Hajian 2019) (Sheppard 2017)
- Native-like Funktionen: Zugriff auf Hardware wie zum Beispiel die Kamera (Hajian 2019)

Bei der Nutzung bzw. der Entwicklung einer PWA entstehen viele Vorteile für die Entwickler und Nutzer. Webtechnologien sind weit verbreitet und für jedes Betriebssystem sind moderne Browser verfügbar, dadurch kann die Anwendung sehr leicht an eine große Nutzergruppe ausgeliefert werden (Rojas 2020). Das Ausliefern an die Nutzer ist auch sehr einfach, weil diese nur eine URL benötigen (Khan, Al-Badi und Al-Kindi 2019) und eine neue Version der App automatisch beim Start der App heruntergeladen wird (Rojas 2020). Außerdem ist die Entwicklung mit Webtechnologien weit verbreitet, man findet viele Ressourcen und Tools, die bei der Entwicklung helfen (Rojas 2020). Beim Vergleichen von nativen Apps zu einer PWA fällt zudem auf, dass die PWA sehr wenig Speicherplatz nach der Installation benötigt (Biørn-Hansen, Majchrzak und Grønli 2017) (Khan, Al-Badi und Al-Kindi 2019). Die Geschwindigkeit einer PWA auf Android kann im Vergleich zu anderen Cross-Platform Ansätzen mithalten oder ist sogar schneller (Biørn-Hansen, Majchrzak und Grønli 2017).

Eine PWA hat nicht nur Vorteile, sondern auch Nachteile. Einer davon ist die Limitierung auf die Funktionen die der Webbrowser zur Verfügung stellt. Hinzu kommt, dass einige Funktionen in den Webbrowsern noch nicht standardisiert sind und / oder noch nicht von allen gängigen Browsern unterstützt werden (Majchrzak, Biørn-Hansen und Grønli 2018) (Biørn-Hansen, Majchrzak und Grønli 2017). Wenn die Performance der App eine große Rolle spielt, wie etwa bei Spielen, ist eine PWA nicht sehr gut geeignet (Biørn-Hansen, Majchrzak und Grønli 2017).

Damit aus einer Website eine PWA wird sind mindestens zwei Dinge erforderlich. Erstens wird eine Manifest Datei benötigt, die Informationen über das Icon, den Namen und vielen mehr enthält (Hajian 2019) (Rojas 2020). Als zweites wird ein Service Worker benötigt, der unter anderem für die Offline-Fähigkeit verantwortlich ist (Rojas 2020).

2.2 Angular und Typescript

Angular ist ein Framework zum Entwickeln von Webanwendungen für alle Plattformen (Google LLC 2020). Es ist Open-Source, getrieben von einer großen Community aber auch von Firmen wie Google weiterentwickelt und selbst viel genutzt (Google LLC 2020). Als Programmiersprache ist Typescript vorgesehen.

Typescript erweitert JavaScript um einige Funktionen wie zum Beispiel Typisierung (Microsoft 2020b). Typescript kann vor dem Ausliefern in reines JavaScript kompiliert werden und ist somit in allen gängigen Browsern oder auch auf Servern mit Node.js ausführbar (Microsoft 2020b).

Die Kombination aus Angular und Typescript bietet eine sehr gute Basis zum Entwickeln von großen, schnellen und skalierbaren Anwendungen (Google LLC 2020). Durch die statische Typisierung sind zum Beispiel Refactorings mit den vielen verfügbaren Tools schnell und sicher durchzuführen (Microsoft 2020b) (Google LLC 2020). In der großen Community findet man auf sehr viele Fragen sofort eine Antwort.

2.3 CROSSLOAD

CROSSLOAD ist ein Webportal das christliche Medien, im Moment hauptsächlich Predigten, zur Verfügung stellt (biblepool gUG 2020). Diese Medien können durchsucht, angehört und heruntergeladen werden. Die Entwicklung befindet sich im Moment in der Beta-Phase (https://beta.crossload.org) und wird vor allem von Ehrenamtlichen vorangetrieben. In Abbildung 2.1 ist die mobile Seite mit den neusten Inhalten auf CROSSLOAD zu sehen. Abbildung 2.1 zeigt die Detailansicht einer Predigt.

Im Frontend wird Angular mit Typescript zum Entwickeln verwendet und das Backend besteht aus vielen verschiedenen Services, die mit unterschiedlichen Sprachen und Frameworks entwickelt werden. Für diese Arbeit ist nur die Technologie für das Frontend interessant. Das Webportal besitzt schon über eine Manifest-Datei und kann somit auf den Startbildschirm des Smartphones hinzugefügt werden. Statische Inhalte werden bereits gecached und sind somit auch offline Verfügbar.

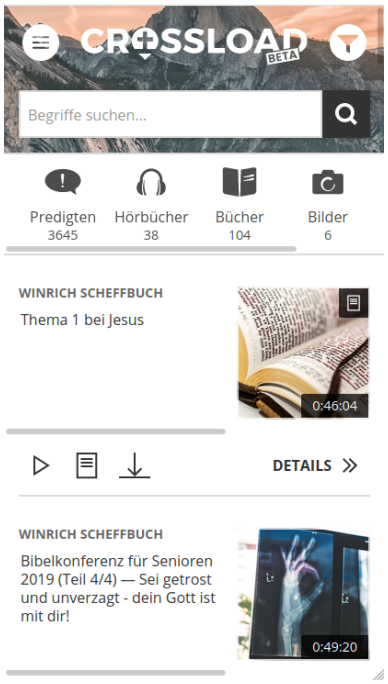

Abbildung 2.1: CROSSLOAD - Mobile Ansicht der neusten Inhalte

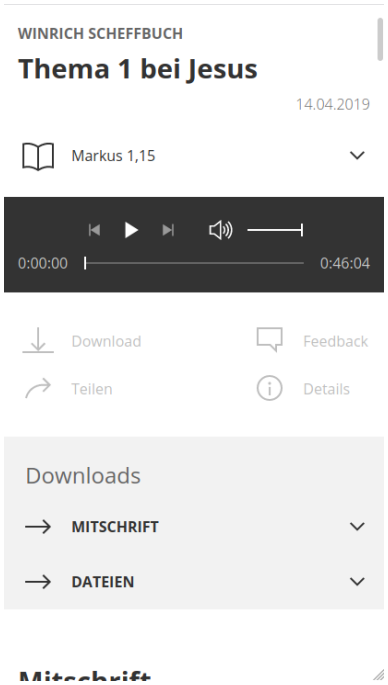

Abbildung 2.2: CROSSLOAD - Dateilansicht einer Predigt

Kapitel 3

Anforderungsanalyse

Die Anforderungen die in diesem Kapitel herausgearbeitet werden, bilden die Grundlage für die Entscheidungen in den nächsten Kapiteln. Zuerst werden grundlegende Bedingungen und die Vorgehensweise erklärt. Anschließend werden einige Szenarien zur Benutzung der Offline Funktion beschrieben und daraus konkrete Anforderungen hergeleitet.

3.1 Vorgehensweise und Randbedingungen

Da CROSSLOAD fast nur ehrenamtliche Mitarbeiter hat, gibt es niemanden der genaue Anforderungen an Features vorschreibt. Vielmehr werden die Anforderungen zusammen im Team diskutiert. Auch für die Offline Funktionalitäten gab es ein Brainstorming mit Mitarbeitern von CROSSLOAD. Daraus haben sich einige Anforderungen herausgestellt. Zusätzlich werden sich einige Szenarien überlegt, die verschiedene Nutzungen der Offline Funktionialität beleuchtet. Die Szenarien helfen beim Verstehen der Anforderungen.

Es gibt zwei Randbedingungen, die für diese Thesis relevant sind. Erstens muss das Framework Angular benutzt werden, weil das bisherige Webportal auch in Angular geschrieben ist. Eine andere Technologie zu verwenden, würde die Integration in die bestehende Platform sehr schwer gestalten. Als zweites stellt sich die Frage welche Browser unterstützt werden müssen. CROSSLOAD befindet sich in der Beta-Phase, deswegen gibt es noch nicht genügend Daten darüber, welche Browser von den Nutzern von CROSSLOAD verwendet werden. Als Anhaltspunkt dient deswegen die durchschnittliche Browsernutzung in Deutschland. Dabei kann zwischen der

Nutzung der Browser auf PCs wie in Abbildung 3.1 zu sehen und auf Mobilgeräten wie in Abbildung 3.2 zu sehen unterschieden werden.

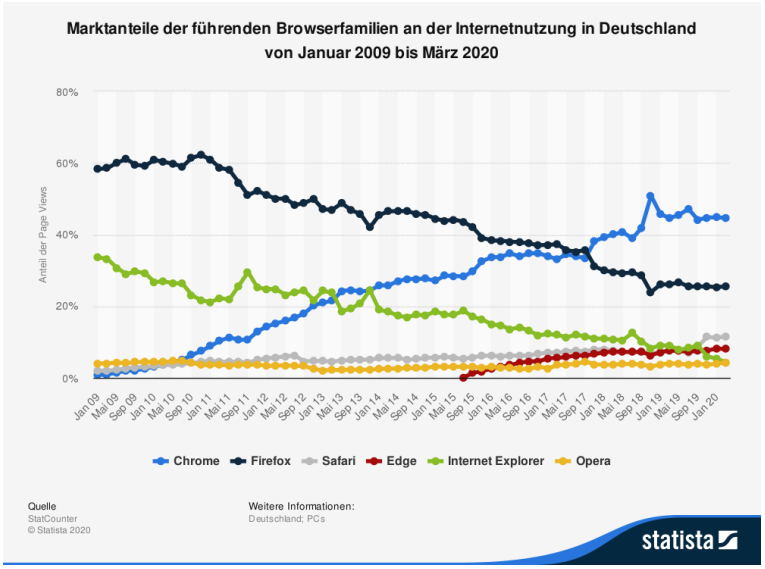

Abbildung 3.1: Internetnutzung nach Browsern in Deutschland (StatCounter 2020a)

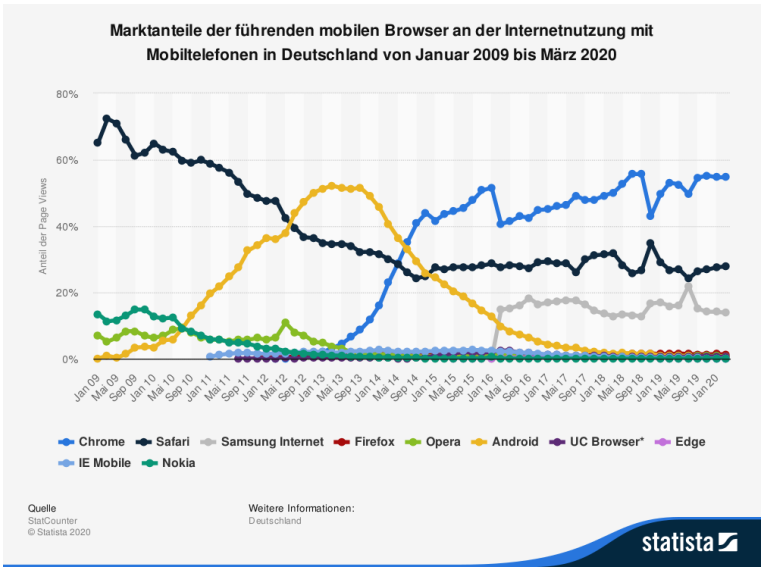

Abbildung 3.2: Internetnutzung nach mobilen Browsern in Deutschland (StatCounter 2020b)

Die Offline-Funktionen für CROSSLOAD, die in dieser Thesis erarbeitet werden, sind nicht essentiel zum Benutzen des Portals. Deswegen kann auch in kauf genommen werden, dass manche Funktionen für Nutzer mit gewissen Browsern nicht zur Verfügung stehen. Natürlich ist es besser, wenn viele Nutzer von den neuen Funktionalitäten profitieren. Wichtig ist nur, dass iOS und Android unterstützt werden.

Dabei wird nicht berücksichtigt, welche Browserversion die Nutzer verwenden. Alle gängingen Browser verfügen über eine automatische Updatefunktion, dadurch erhalten die meisten Nutzer sehr zeitnah veröffentlichte Updates.

Bei den Browsern für Mobilgeräte ist der Markanteil von Chrome, Safari und Samsung Internet zusammengerechnet über 96%. Außerdem basiert Samsung Internet intern auf Chromium und teilt somit die allermeisten Funktionen mit Chrome (Appelquist 2019). In späteren Entscheidungen wird deswegen Samsung-Internet nur erwähnt wenn es von der Funktionalität im Chrome Browser abweicht.

Für PCs gibt es mehr Browserfamilien, die relevant sein könnten. Insgesamt lassen sich alle relevanten Browser auf drei Gruppen reduzieren: Safari und Firefox haben jeweils eine eigene Render-Engine. Opera setzt schon seit einigen Jahren auf Chromium auf (Opera Software 2013) und auch für den Edge-Browser wurde 2018 angekündigt Chromium als Basis zu verwenden (Microsoft 2020a). Der Internet Explorer wird immer weniger verwendet und von Microsoft nicht mehr für den normalen Endnutzer empfohlen (Microsoft 2020a), deswegen wird der Internet Explorer hier auch nicht berücksichtigt. In den nachfolgenden Kapiteln wird der Browser Edge und Opera mit Chrome gleichgesetzt und nicht extra erwähnt, solange keine relevanten Unterschiede vorhanden sind. Die Browser Safari, Firefox und Chrome (inklusive der Chromium basierten Browser) haben einen Marktanteil von über 94%.

3.2 Szenarien

Jedes Szenario beschreibt eine konkrete Ineraktion mit dem Webportal ohne Sonderfälle abzubilden. Sie dienen dazu die Anforderungen besser zu verstehen. Im folgenden werden drei Szenarien beschrieben.

3.2.1 Predigt im Auto anhören

Akteur: Benutzer

Ablauf:

1. Benutzer ist zu Hause und durchsucht Inhalte auf CROSSLOAD
2. Benutzer favorisiert sich mehrere Inhalte

3. Der Download der Inhalte startet
4. Sobald der Download fertig ist, wird das dem Benutzer angezeigt
5. Benutzer geht außer Haus in sein Auto
6. Benutzer bekommt alle favorisierten Inhalte angezeigt
7. Benutzer wählt einen favorisierten Inhalt aus und sieht die Übersichtsseite des Inhalt
8. Benutzer hört sich die Predigt an und benötigt dafür kein Internet

3.2.2 Inhalt für die Reise vormerken

Akteur: Benutzer

Ablauf:

1. Benutzer ist am Flughafen und steht kurz vor einem Flug
2. Benutzer favorisiert sich einen Inhalt auf CROSSLOAD
3. CROSSLOAD fragt den Benutzer, ob er den Inhalt auch über mobiles Inernet herunterladen möchte
4. Der Benutzer bestätigt diese Anfrage
5. Der Download beginnt
6. Sobald der Download fertig ist, wird das dem Benutzer angezeigt
7. Benutzer ist im Flugzeug und hört sich die favorisierte Predigt an

3.2.3 Inhalt herunterladen sobald eine WLAN Verbindung besteht

Akteur: Benutzer

Ablauf:

1. Benutzer ist unterwegs und bekommt einen Inhalt auf CROSSLOAD empfohlen
2. Benutzer favorisiert diesen Inahlt

3. CROSSLOAD fragt den Benutzer, ob er den Inhalt auch über mobiles Inernet herunterladen möchte
4. Der Benutzer verneint diese Anfrage
5. Der Benutzer kommt nach Hause und ist mit dem eigenen WLAN verbunden
6. Der Download der favorisierten Predigt beginnt
7. Sobald der Download fertig ist, wird das dem Benutzer angezeigt
8. Der Benutzer hört sich die favorisierte Predigt an. Obwohl er eine WLAN-Verbindung besitzt, werden die heruntergeladenen Inhalte zum Abspielen der Predigt genutzt

3.3 Anforderungen

Aus den Diskussionen mit Mitarbeitern von CROSSLOAD haben sich die Anforderungen herausgestellt, die in diesem Kapitel aufgelistet sind. Für diese Thesis sind nur Audioinhalte auf CROSSLOAD relevant.

- Der Benutzer kann sich einen Inhalt vormerken
- Der vorgemerkte Inhalt wird automatisch anhand der Verbindungsart heruntergeladen. Der genaue Ablauf ist in einem Aktivitätsdiagramm in Abbildung 3.3 zu sehen.
  - Wenn eine WLAN-Verbindung besteht wird der Download sofort gestartet
  - Wenn eine mobile Datenverbindung besteht wird der Benutzer gefragt, ob er den Inhalt jetzt herunterladen möchte
  - Wenn der Benutzer dies verneint wird auf eine WLAN-Verbindung gewartet und der Download gestartet, sobald diese besteht
- Der Benutzer wird über den Fortschritt des Downloads informiert
- Das Portal erkennt wenn ein Benutzer keine Internetverbindung hat und leitet ihn auf eine spezielle Seite weiter
- Die zurzeit heruntergeladenen Inhalte können in einer Übersicht gesehen werden

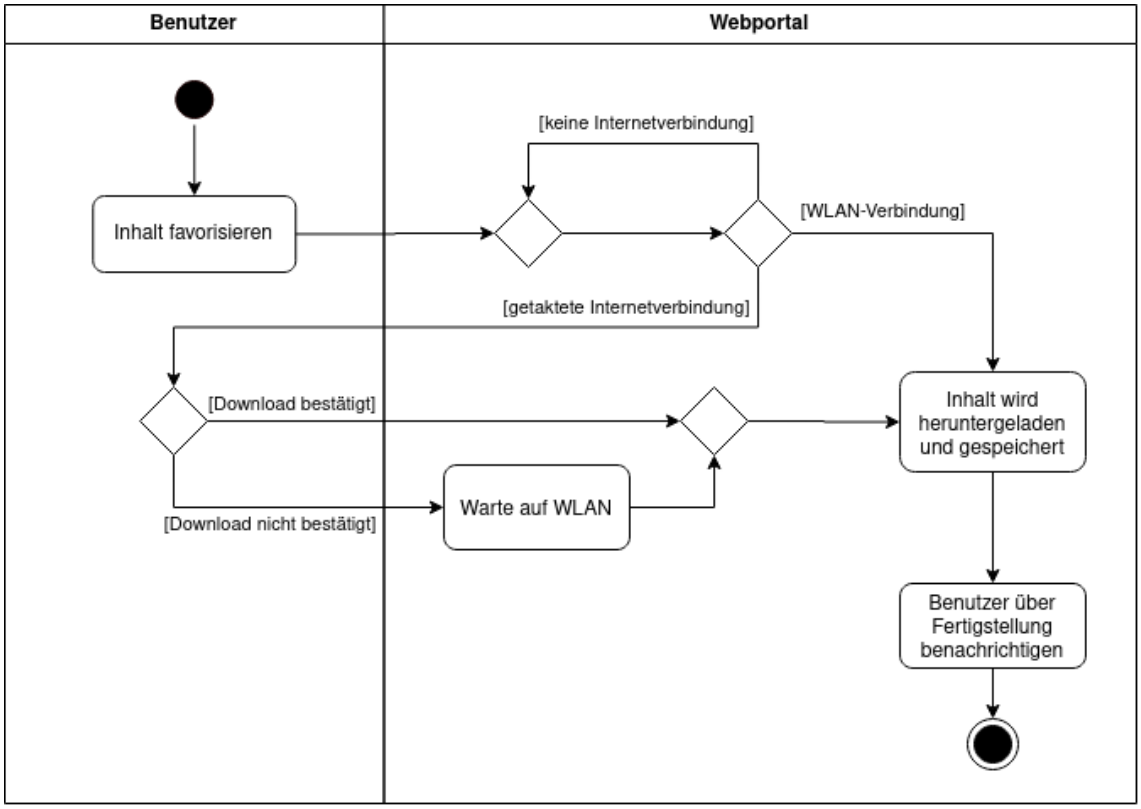

Abbildung 3.3: Aktivitätsdiagramm: Download des Inhalt abhängig von der Verbindungsart

- Die zurzeit heruntergaladenen Inhalte können gelöscht werden
- Es soll möglich sein, für einen bestimmten Inhalt herauszufinden, ob dieser Offline verfügbar ist
- Die Inhalte der Detailseite eines heruntergeladenen Inhalts werden gespeichert
- Der heruntergeladene Inhalt kann angehört werden
  - Wenn ein Inhalt angehört werden soll, der bereits heruntergeladen ist, sollen die heruntergeladenen Daten verwendet werden, um Traffic zu vermeiden

Kapitel 4

Konzeption

4.1 Lokale Datenspeicherung

Moderne Webbrowser stellen viele verschiedene Lösungen zum lokalen Speichern von Daten zu Verfügung. In diesem Kapitel werden einige davon mit ihren Stärken und Schwächen vorgestellt. Zuletzt wird eine passende Lösung für CROSSLOAD ausgewählt und begründet.

4.1.1 Web Storage

Der Web Storage besitzt zwei unterschiedliche Arten Daten zu speichern: LocalStorage und SessionStorage (Hajian 2019) (*Web Storage API* 2020). Die beiden Speicher unterscheiden sich nur in der Dauer der Speicherung der Daten. Die Daten vom SessionStorage werden gelöscht sobald die Sitzung auf der Website vorbei ist, das heißt der Browser oder der Tab geschlossen wird (Hajian 2019) (*Web Storage API* 2020). Der LocalStorage bleibt über mehrere Sitzungen erhalten und wird nicht automatisch gelöscht (Hajian 2019) (*Web Storage API* 2020).

Die Web Storage API bietet die Möglichkeit Schlüssel / Wert Paare im Browser abzulegen (*Web Storage API* 2020). Alle relevanten Browser implementieren diese API (*Web Storage API* 2020), es gibt aber auch einige Einschränkungen:

- Zugriffe sind nur synchron möglich (Hajian 2019)
- Als Schlüssel und Werte können jeweiles nur Strings gespeichert werden (Hajian 2019)

- Die API ist nicht von Web Workern aufrufbar (Hajian 2019). Service Worker haben zum Beispiel keinen Zugriff darauf
- Es können maximal 5 Megabyte (MB) an Daten gespeichert werden (*Web Storage API* 2020)

Aufgrund der synchronität und des geringen Datenvolumes das gespeichert werden darf, eignet sich die Web Storage API nur zum Speichern von wenig Daten. Größere Daten und insbesondere binäre Dateien, die nicht sehr gut in eine textuelle Form übertragen werden können sollten nicht abgespeichert werden.

4.1.2 File System API

Die File System API und FileWrite API bieten dem Browser die Möglichkeit Dateien in ein virtuelles Dateisystem abzulegen und von dort wieder zu laden (Hajian 2019) (LePage 2020). Dieses virtuelle Dateisystem unterstützt viele gängige Funktionen wie Ordner und Dateimanipulationen, wie man sie von herkömmlichen Dateisystem gewohnt ist (*Introduction to the File and Directory Entries API* 2019). MDN nennt diese API File and Directory Entries API (*Introduction to the File and Directory Entries API* 2019).

Der Vorteil dieser API ist der Umgang mit binary large objects (blobs), wie zum Beispiel Audiodateien. Diese können leicht gespeichert, geladen und manipuliert werden (*Introduction to the File and Directory Entries API* 2019). Außerdem gibt es eine API für synchrone und asynchrone Zugriffe und ist ebenso in Web Workern verfügbar (Hajian 2019).

Es gibt keinen offiziellen Standard (*Introduction to the File and Directory Entries API* 2019) (*Filesystem & FileWriter API* 2020). Deswegen unterstützen noch nicht sehr viele Browser diese API, bisher ist in Chrome und in allen Chromium basierten Browser diese Funktion verfügbar (*Filesystem & FileWriter API* 2020).

4.1.3 Web SQL

Web SQL ist eine API, die es erlaubt Daten in einer Datenbank zu speichern und diese Daten mit einer SQL ähnlichen Sprache zu durchsuchen (Hickson 2010). Die

Anfragen sind asynchron funktionieren aber nicht in einem Web Worker (Hajian 2019).

Diese API wurde nie von allen Browsern implementiert und ist mitlerweile deprecated, soll also nicht mehr verwendet werden (Hajian 2019).

Aufgrund der Tatsache, dass Web SQL deprecated ist, wird diese Technologie auch keinen Einsatz bei CROSSLOAD finden. Als Datenbank in Browsern hat sich IndexedDB etabliert.

4.1.4 Indexed DB

- Umfangreiche Objektdatenbank um große Datenmengen auf dem Client zu speichern. Unterstützt Indexe zum Durchsuchen. Unterstützt auch Transaktionen (Sheppard 2017)

- key-value pari NoSQL Datenbank zum Speichern von vielen Daten (bis zu 20-50 % des verfügbaren Speicherplatzes). Unterstützt viele Datentypen, ist asynchron und kann überall verwendet werden (Hajian 2019)

- Verfügbare Bibliotheken: LocalForage, Dexie.js, ... (Hajian 2019)

Transaktionelle Datenbank - Gut für viel Daten, die Durchsucht werden müssen - Asynchron

4.1.5 Cache API

- Service Worker: Laufen im Hintergrund (im eigenen Thread) ohne Zugriff aufs DOM (Sheppard 2017)

- Service Worker ist ein Mittler zwischen APP und Internet. Es übernimmt Aufgaben wie Caching, Syncing, Benachrichtigungen (Sheppard 2017)

Service Worker: - Ärbeiterïn JavaScript die im Hintergrund laufen - Ermöglicht es Netzwerk-Request zu unterbrechen (Proxy) - Benachrichtigungen zu senden - Cache Verwalten und vieles mehr - Laufen im eigenen Thread ohne DOM Zugriff, HTTPS-Only - Website sollte nicht auf Service Worker angewiesen sein (Hajian 2019)

Angular bietet einen einfachen Weg Service Workers zu verwenden: - ng add @angular/pwa (Hajian 2019)

Cache Strategien: - Cache only: Es wird nur der Inhalt des Caches geliefert. Wenn nicht verfügbar wird nichts geliefert - Netzwerk only: Es wird nur die Antwort des Requests gesendet - Cache-First: Cache und wenn nicht verfügbar -> Netzwerk - Netzwerk-First: Netzwerk und wenn nicht verfügbar -> Cache - Cache und Netzwerk (stale-while-revalidate): Nimm erst den Wert vom Cache und sobald das Netzwerk geantwortet hat, die Antwort vom Netzwerk. Gut für sich häufig ändernde Daten (Facebook) - Generic-Fallback: Zum Beispiel für ein Bild ein Standardbild liefern, wenn Netzwerk und Cache keins haben (Hajian 2019) (Rojas 2020)

Bibliothek für Service Worker: - Workbox (Rojas 2020)

Cache storage != browser cache (Rojas 2020)

4.1.6 Überblick und Entscheidung

In Tabelle 4.1 werden die vorgestellten Möglichkeiten zum Speichern von Daten nach folgenden Kriterien verglichen: maximale Datenmenge, mögliche Datentypen, synchron / asynchron, Browsersupport, in Web Worker aufrufbar. Bei fehlenden Feldern in Web SQL wurde nicht weiter recherchiert, weil diese Technologie veraltet ist und nicht mehr verwendet werden soll.

Tabelle 4.1: Vergleich der APIs zur lokalen Datenspeicherung

Technologie	Datenmenge	Datentypen	(a)synchron	Browsersupport	Web Worker	Anmerk
Web Storage API	max. 5 MB	String	nur synchron	alle	nein	
File System API	quota	blobs und weitere	synchron und asynchron	nur Chrome	Ja	
Web SQL	-	.	synchron und asynchron	teilweise	nein	veraltet
Indexed DB						
Cache API						

4.2 Herunterladen der Audiodaten

4.2.1 Fetch API

Einfachere XMLHttpRequests, Promise basierend (Rojas 2020)

4.2.2 Background Fetch

4.2.3 Background Sync

- Speichert den API Call bis eine stabile Internetverbindung besteht und schickt ihn dann raus. Sogar wenn die APP nicht aktiv ist oder läuft (Sheppard 2017)

- Versendet Request sobald eine Internetverbindung besteht, auch wenn die APP nicht offen ist (Rojas 2020)

- Man kann diese API mit IndexedDB kombinieren. Daten können in der Datenbank zwischengespeichert werden, bis eine Verbindung besteht (Rojas 2020)

4.3 Verbindungsstaus auslesen

- navigator.onLine: ist nicht 100% genau. Wird manchmal sagen, dass das Gerät online ist wenn es nur mit einem Netzwerk verbunden ist (Sheppard 2017)

Kapitel 5

Implementierung

5.1 Grafische Oberfläche

5.2 Lokale Datenspeicherung

5.3 Herunterladen der Audiodaten

Kapitel 6

Evaluation und Reflexion

6.1 Kann durch die PWA eine native APP ersetzt werden?

6.2 Nutzerfreundlichkeit

Kapitel 7

Zusammenfassung und Ausblick

7.1 Zusammenfassung

7.2 Ausblick

- Bei anderen Nutzern wurde beobachtet, dass Predigten nicht manuell gelöscht werdne, wenn man die mp3-Dateien heruntergealden hat. Evtl. automatisches Löschen angehörtet Predigten

- Heruntergeladene Inhalte sortieren / filtern / suchen

- Die Favoriten sollen über einen Account synchronisiert werden können

Abkürzungsverzeichnis

API Application Programming Interface
MB Megabyte
blob binary large object
PWA Progressive Web App

Tabellenverzeichnis

Abbildungsverzeichnis

Literatur

Appelquist, Daniel (9. Okt. 2019). *Samsung Internet 10.2 Beta*. URL: https://medium.com/samsung-internet-dev/samsung-internet-10-2-beta-d741ea15906d (besucht am 23. 05. 2020).

biblepool gUG (2020). *Crossload - Deine Tankstelle für Wachstum im Glauben*. URL: https://crossload.org/ (besucht am 20. 05. 2020).

Biørn-Hansen, Andreas, Tim A Majchrzak und Tor-Morten Grønli (2017). „Progressive Web Apps: The Possible Web-native Unifier for Mobile Development." In: *WEBIST*, S. 344–351.

Filesystem & FileWriter API (2020). URL: https://caniuse.com/#feat=filesystem (besucht am 29. 05. 2020).

Google LLC (2020). *Angular*. URL: https://angular.io/ (besucht am 20. 05. 2020).

Hajian, Majid (2019). „Safety Service Worker". In: *Progressive Web Apps with Angular: Create Responsive, Fast and Reliable PWAs Using Angular*. Berkeley, CA: Apress, S. 283–288. DOI: 10.1007/978-1-4842-4448-7_11. URL: https://doi.org/10.1007/978-1-4842-4448-7_11.

Hickson, Ian (18. Nov. 2010). *Web SQL Database*. URL: https://www.w3.org/TR/webdatabase/ (besucht am 29. 05. 2020).

Introduction to the File and Directory Entries API (24. Sep. 2019). URL: https://developer.mozilla.org/en-US/docs/Web/API/File_and_Directory_Entries_API/Introduction (besucht am 29. 05. 2020).

Khan, Asharul Islam, Ali Al-Badi und Mahmood Al-Kindi (2019). „Progressive Web Application Assessment Using AHP". In: *Procedia Computer Science* 155. The 16th International Conference on Mobile Systems and Pervasive Computing (MobiSPC 2019),The 14th International Conference on Future Networks and Communications (FNC-2019),The 9th International Conference on Sustainable Energy Information Technology, S. 289–294. DOI:

https://doi.org/10.1016/j.procs.2019.08.041. URL: http://www.sciencedirect.com/science/article/pii/S187705091930955X.

LePage, Pete (27. Apr. 2020). *Storage for the web*. URL: https://web.dev/storage-for-the-web/ (besucht am 29. 05. 2020).

Majchrzak, Tim A, Andreas Biørn-Hansen und Tor-Morten Grønli (2018). „Progressive web apps: the definite approach to cross-platform development?“ In:

Microsoft (6. März 2020a). *Häufig gestellte Fragen (FAQs) für IT-Experten*. URL: https://docs.microsoft.com/de-de/microsoft-edge/deploy/microsoft-edge-faq (besucht am 23. 05. 2020).

— (2020b). *Typescript - JavaScript that scales.* URL: https://www.typescriptlang.org/ (besucht am 20. 05. 2020).

Opera Software (13. Feb. 2013). *Opera gears up at 300 million users*. URL: https://press.opera.com/2013/02/13/opera-gears-up-at-300-million-users/ (besucht am 23. 05. 2020).

Rojas, Carlos (2020). „Service Workers“. In: *Building Progressive Web Applications with Vue.js : Reliable, Fast, and Engaging Apps with Vue.js*. Berkeley, CA: Apress, S. 55–65. DOI: 10.1007/978-1-4842-5334-2_3. URL: https://doi.org/10.1007/978-1-4842-5334-2_3.

Sheppard, Dennis (2017). „Background Sync for Offline Apps with Service Workers“. In: *Beginning Progressive Web App Development: Creating a Native App Experience on the Web*. Berkeley, CA: Apress, S. 73–93. DOI: 10.1007/978-1-4842-3090-9_5. URL: https://doi.org/10.1007/978-1-4842-3090-9_5.

StatCounter (24. Apr. 2020a). *Marktanteile der führenden Browserfamilien an der Internetnutzung in Deutschland von Januar 2009 bis März 2020*. URL: https://de.statista.com/statistik/daten/studie/13007/umfrage/marktanteile-der-browser-bei-der-internetnutzung-in-deutschland-seit-2009/ (besucht am 09. 05. 2020).

— (24. Apr. 2020b). *Marktanteile der führenden mobilen Browser an der Internetnutzung mit Mobiltelefonen in Deutschland von Januar 2009 bis März 2020*. URL: https://de.statista.com/statistik/daten/studie/184297/umfrage/marktanteile-mobiler-browser-bei-der-internetnutzung-in-deutschland-seit-2009/ (besucht am 09. 05. 2020).

Web Storage API (8. Feb. 2020). URL:
https://developer.mozilla.org/en-US/docs/Web/API/Web_Storage_API
(besucht am 29. 05. 2020).