

Progressive Web App

Studienarbeit

des Studiengangs IT Automotive
an der Dualen Hochschule Baden-Württemberg Stuttgart

von

Emmelie Beitlich, Finn Freiheit

Oktober 2020

Bearbeitungszeitraum
Matrikelnummer, Kurs
Gutachter

12 Wochen
2533282, ITA19
Dipl.-Ing. (FH) Peter Pan

Erklärung

Wir versichern hiermit, dass wir unsere Studienarbeit mit dem Thema: *Progressive Web App* selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt haben. Wir versichern zudem, dass die eingereichte elektronische Fassung mit der gedruckten Fassung übereinstimmt.

Stuttgart, Oktober 2020

Emmelie Beitlich, Finn Freiheit

Abstract

Inhaltsverzeichnis

Abkürzungsverzeichnis	V
Abbildungsverzeichnis	VI
Tabellenverzeichnis	VII
Listings	VIII
1. Einleitung	1
1.1. Motivation	1
1.2. Begriffsklärung	1
1.3. Aufbau der Arbeit	3
I. Theoretische Grundlagen	4
2. Architektur	5
3. Angular	7
3.1. Strukturdirektiven	7
3.2. Interpolation	8
3.3. Property Binding	8
3.4. Event Binding	8
3.5. Backend Anbindung	9
4. Node.js	10
5. Datenbanken	11
5.1. SQL	11
5.1.1. ACID	11
5.2. NoSQL	11
5.3. SQL vs NoSQL	12
6. MongoDB	14
6.1. Überblick	14
6.2. Speicherung der Daten	14
6.2.1. JSON-basierte Speicherung	14
6.2.2. BSON	15
6.2.3. Identifier	15

7. Progressive Web App	16
7.1. Web-App-Manifest	16
7.1.1. short_name	17
7.1.2. icons	17
7.1.3. start_url	17
7.1.4. display	17
7.1.5. theme_color	18
7.1.6. Debugging vom Manifest	18
7.2. Service Workers	19
7.2.1. Der Lebenszyklus eines Service Workers	19
7.3. Push-Notifikations	25
7.3.1. Push-Registrierung	25
7.4. Progressive Web Apps in Angular	27
7.5. PWA für IOS	28
7.6. Endgerät Emulation	29
Literatur	32
Anhang	35
.1. HTML	35
.1.1. Attribute	36
.2. CSS	36
.2.1. CSS in die HTML-Datei einbinden	37
.3. JavaScript	38
.3.1. JavaScript in die HTML-Datei einbinden	38

Abkürzungsverzeichnis

PWAs	Progressive Web Apps
PWA	Progressive Web App
JSON	JavaScript Object Notation
URL	Uniform Resource Locator
API	Application Programming Interface
DOM	Document Object Model
SPA	Single Page Application
SQL	Structured Query Language
CRUD	create, read, update and delete
HTTP	Hypertext Transfer Protocol
HTML	Hyper Text Markup Language
CSS	Cascading Style Sheets
REST	Representational State Transfer
NoSQL	Not only SQL
CLI	Command Line Interface
AVD Manager	Android Virtual Device Manager
IETF	Internet Engineering Task Force
VAPID	Voluntary Application Server Identification

Abbildungsverzeichnis

2.1. Architektur der gesamten Anwendung	5
7.1. display in Standalone Einstellung	17
7.2. display in Minimal User Interface Einstellung	17
7.3. Theme Color auf Weiß geändert	18
7.4. Entwicklereinstellungen Web-App-Manifest	18
7.5. Google Chrome Developer Tools zum einsehen der im Cache gespeicherten Dateien.	22
7.6. Laden einer Applikation ohne Service Worker	23
7.7. Laden einer Applikation ohne Internetverbindung	24
7.8. Sequenzdiagramm für die Ereignisse einer Anmeldung, push Nachrichttransfer und Abmeldung [BT16]	26
7.9. Überprüfung der Pushnachricht Funktionalitäten in den Browsern(a) Google Chrome , (b) Apple Safari	28
7.10. (a) Emulierte Android Handy in Android Studio, (b) PWA im Browser geöffnet	30
7.11. (a) Installation der PWA unter Android, (b) Installierte PWA	31
.12. HTML Grundgerüst	36
.13. Grüne Überschrift	37

Tabellenverzeichnis

1.1. Plattformen und die dazu benötigten Programmiersprachen	1
--	---

Listings

3.1. Interpolation im Template	8
3.2. Property Binding	8
3.3. Event Binding	9
7.1. Registrierung des Service Workers	20
7.2. Angular <i>ngsw-config.json</i> - Datei zu Angabe der Ressourcen, die durch den Service Worker in den Cache gespeichert werden sollen.	21
7.3. Push-Registrierung unter Berücksichtigung der Konzepte von Progressive Enhancement	27
4. Grundgerüst einer HTML-Seite	35
5. HTML Attribute	36
6. Die generelle Syntax für CSS-Eigenschaften	36
7. Grüne Überschrift CSS	37
8. CSS-Datei in HTML verlinken	37
9. CSS-Datei in HTML einbinden	38
10. JavaScript in HTML einbinden	39

1. Einleitung

1.1. Motivation

Folgendes Szenario soll ein Einblick in die Vorteile von Progressive Web Apps ([PWAs](#)) aufzeigen.

Ein junges Startup aus IT-Studenten hat eine Idee für eine Applikation. Ihr Ziel ist es diese Applikation an so viele Nutzer wie möglich zu verbreiten. Die Applikation soll daher für folgende Plattformen, siehe Tabelle ?? erhältlich sein.

Plattform	Programmiersprache
IOS	Swift
Android	Java oder Kotlin
MacOS	Swift
native Windows	C++ oder C#
Webbrowser	JavaScript, HTML und CSS

Tabelle 1.1.: Plattformen und die dazu benötigten Programmiersprachen

Wie man anhand der Tabelle sehen kann, wird eine Vielzahl an unterschiedlichen Programmiersprachen benötigt, um die Applikation über mehrere Plattformen zu verbreiten. Das junge Startup verfügt leider nicht über die Kapazitäten um die Applikation in jeder dieser Programmiersprachen zu implementieren und zu warten.

Aus diesem Grund entscheidet sich das Startup dafür eine Progressive Web App ([PWA](#)) zu erstellen. Eine PWA ist eine Webanwendung mit erweiterten Funktionen. Die Besonderheit dieser erweiterten Webanwendung liegt darin, dass sie einmal implementiert auf sämtliche Plattformen installiert werden kann. Sie ist somit Plattform unabhängig.

1.2. Begriffsklärung

der Begriff *Progrssive Web App* setzt sich aus den Begriffen *Web App* und *Progressive Enhancement* zusammen. Eine Web App (deutsch Webanwendung) ist eine mithilfe von

JavaScript, HTML und CSS entwickelte Applikation. Der zweite Begriff wurde von Steve Champeon im Jahre 2003 in seiner Publikation mit dem Titel *progressive enhancement and the future of web design* geprägt [Cha].

Unter dem Begriff *Progressive Enhancement* (deutsch Progressive Verbesserung) verbirgt sich das Ziel Webseiten so zur Verfügung zu stellen, dass jeder Webbrower in der Lage ist, die grundlegendste Form einer Webseite dazustellen. Hierbei ist es unabhängig über welche Version der Browser oder das Endgerät verfügt. Alle zusätzlichen Funktionalitäten, die eventuell erst mit modernen Browsern und Endgeräten genutzt werden können, werden erst im anschluss in form von Skripten eingebunden.

Um PWAs nutzen zu können werden die neusten Funktionen der modernen Webbrowser benötigt, darunter *service workers* und *web app manifests* (Referenz Kapitel).

Google hat das Konzept von PWAs im Jahr 2015 vorgestellt und ist seit dem maßgeblich an der Entwicklung beteiligt. Das Ziel bei der Entwicklung von PWAs liegt darin die Vorteile von Nativen Applikation mit den Vorteilen von Webanwendung zu kombinieren.

Native Applikation beziehungsweise Plattformspezifische Applikation sind sehr Funktionsreich und zuverlässig. Weitere Vorteile sind, das sie :

- Netzwerkunabhängig funktionieren,
- lokale Dateien aus dem Dateisystem lesen und schreiben können,
- auf Hardwareschnittstellen wie USB und bluetooth zugreifen können,
- mit Daten des Gerätes interagieren können, wie zum Beispiel Fotos oder aktuell spielende Musik.

Webapplikationen wiederum sind sehr gut erreichbar, sie können verlinkt, über Suchmaschinen gefunden und geteilt werden.

Mithilfe von PWAs können Applikation erzeugt werden, die:

- Installierbar sind, Kapitel,
- auf Geräteschnittstellen zugreifen können, Kapitel
- Netzwerkunabhängig funktionieren, Kapitel
- Push-Notifikations versenden können, Kapitel

PWAs sind somit laut Sam Richard und Pete LePage das beste aus zwei Welten [Sam]. Mithilfe von progressiver Verbesserung werden die modernen Funktionen von Browsern genutzt um die Vorteile von Plattformspezifischen Anwendungen nutzen zu können. Sind die dafür benötigten Funktionen wie zum Beispiel *service workers* nicht vorhanden, können dennoch die Grundfunktionen der Anwendung im Web genutzt werden.

1.3. Aufbau der Arbeit

Teil I.

Theoretische Grundlagen

2. Architektur

Die grundlegende Architektur einer Webanwendung ist in Abbildung 2.1 beschrieben. Man spricht von einer Client-Server-Architektur. Die Nutzer einer Webanwendung interagieren mit einem Browser (dem Client). In dem Browser wird die Webseite (das Frontend, Kapitel 3) dargestellt. Die Inhalte einer Webseite werden mithilfe von Hyper Text Markup Language ([HTML](#)) (siehe Abschnitt .1) und das Layout mithilfe von Cascading Style Sheets ([CSS](#)) (siehe Abschnitt .2) definiert. Auf Nutzerinteraktionen kann dynamisch mithilfe von JavaScript reagiert werden. Ein Browser kann also die drei Skript-Sprachen HTML, CSS und JavaScript interpretieren.

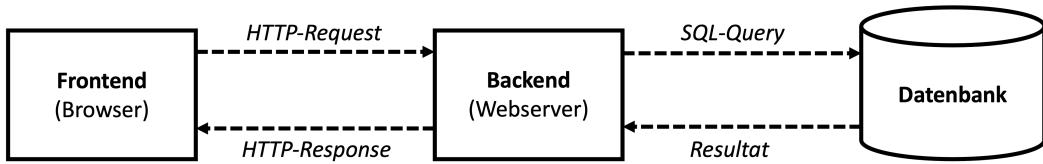


Abbildung 2.1.: Architektur der gesamten Anwendung

Die Webseiten liegen auf einem Webserver (dem Server) bzw. werden von einem Webserver bereitgestellt. In den Browser wird eine Uniform Resource Locator ([URL](#)) eingegeben, welche die Adresse eines Webservers und den Namen der darauf befindlichen Webseite enthält. Die Kommunikation zwischen Browser und Webserver erfolgt mittels Hypertext Transfer Protocol ([HTTP](#)). Die Eingabe einer URL in den Browser erwirkt einen sogenannten *HTTP-Request* an den Webserver. Typischerweise wird eine GET-Anfrage an den Webserver gestellt, um die Webseite zu laden. Der Webserver beantwortet diesen Request mit einer *Response*, indem der Webserver die angefragte Seite an den Browser sendet. Diese wird im Browser dargestellt. Wird innerhalb der Webseite auf einen Hyperlink geklickt, entspricht das in der Regel einer weiteren Anfrage an den (oder einen anderen) Webserver und eine neue Seite wird übermittelt und dargestellt.

Es kann jedoch sein, dass die angefragte Webseite nicht bereits fertig (*statisch*) auf dem Webserver bereitgestellt ist, sondern eine solche Webseite erst auf dem Webserver *dynamisch* zusammengestellt werden muss. Dies ist z.B. der Fall, wenn Suchanfragen durch den Nutzer gestellt und die Ergebnisse der Suche zunächst aus einer Datenbank extrahiert und dann in eine Webseite eingebunden werden müssen. Der Webserver kommuniziert in einem solchen Fall mit der an den Webserver angebundenen Datenbank mittels Structured Query Language ([SQL](#)). Insbesondere in dem Fall, dass durch den Webserver die an den

Browser zu übertragende Webseite erst „zusammengebaut“ werden muss, spricht man beim Webserver auch vom sogenannten *Backend*.

Die Kommunikation vom Browser an den Webserver beinhaltet jedoch nicht nur solche GET-Anfragen, die eine Ressource (Webseite) vom Webserver anfordern, sondern kann darüber hinaus auch das Senden von Daten an den Webserver beinhalten. Das ist z.B. der Fall, wenn eine Webseite ein Formular enthält, in das Daten eingeben werden können und diese Daten entweder in die Datenbank gespeichert werden sollen oder aber als neue Daten zur Aktualisierung alter Daten in der Datenbank verwendet werden. Neben den GET-Anfragen sind deshalb im HTTP-Standard auch sogenannte POST- (Senden neuer Daten), PUT- (Aktualisieren von Daten) und DELETE- (Löschen von Daten) -Anfragen vorgesehen. Betrachtet man die Möglichkeiten zur Manipulation einer Datenbank, dann gibt es vier verschiedene Operationen, die auf einer Datenbank möglich sind:

- *Create* : das Hinzufügen neuer Daten(sätze),
- *Read* : das Lesen einer oder mehrerer Daten(sätze),
- *Update* : das Aktualisieren einer oder mehrerer Daten(sätze) sowie
- *Delete* : das Löschen einer oder mehrerer Daten(sätze).

Diese vier Operationen werden deshalb auch unter dem Begriff create, read, update and delete (**CRUD**) zusammengefasst. Die oben genannten HTTP-Anfragen lassen sich somit gut auf diese CRUD-Operationen abbilden:

- GET-Anfrage für das Lesen (read),
- POST-Anfrage für das Erstellen (create),
- PUT-Anfrage für das Aktualisieren (update) und
- DELETE-Anfrage für das Löschen (delete)

einer Ressource (Daten). Dieses „Mapping“ bildet die Grundlage für eine Representational State Transfer (**REST**)-Schnittstelle. *REST* stellt eine Beschreibung von konkreten HTTP-Anfragen an konkrete Ressourcen auf dem Webserver (oder der Datenbank) dar. Es verbindet also eine HTTP-Anfrage mit einer Ressource und beschreibt somit eindeutig, was mit dieser Ressource geschehen soll. In der vorliegenden Arbeit wurde eine REST-Schnittstelle mithilfe von *Node.js* implementiert, siehe Abschnitt 4. Mithilfe der Schnittstellen werden Daten der Not only SQL (**NoSQL**)-Datenbank names *MonogDB* manipuliert, siehe Abschnitt 5.

3. Angular

Für die Implementierung der Webanwendung wird ein Framework verwendet. Ein Framework ist ein Programmiergerüst, das verwendet werden kann, um modulare, skalierbare und gut wartbare Applikationen zu entwickeln. In der Webentwicklung ist Angular neben React.js und Vue.js eines der beliebtesten Frameworks [sta21].

Mit Angular werden komponentenbasierte Single Page Applikation (**SPA**) erstellt. Bei einer *Single Page Applikation* wird immer nur eine Seite im Browser geladen. Der Inhalt dieser einen Seite ändert sich je nach Nutzerinteraktion. Dies hat unter anderem einen performanten Vorteil, da nur die benötigten Inhalte berechnet werden müssen. Mithilfe des Frameworks können sehr große Webanwendung entwickelt werden. Um die Übersichtlichkeit zu erhöhen, werden die Funktionen der Anwendung in Komponenten aufgeteilt. Die Komponenten sind die Grundbausteine einer Angular-Anwendung.

Eine Komponente besteht aus einem *Template* und einer *TypeScript-Klasse*. Das Template ist für die Darstellung von Inhalten verantwortlich und besteht aus einer HTML-Datei. Die TypeScript-Klasse verwaltet und manipuliert die Daten, die im Template angezeigt werden. TypeScript ist eine Obermenge von JavaScript und unterstützt eine typsichere und objektorientierte Programmierung [Typ21].

Neben dem Template und der TypeScript-Klasse verfügt eine Komponente über eine Datei, um CSS-Eigenschaften zu deklarieren. Bei der zu erstellenden Angular-Anwendung handelt es sich um *SCSS-Dateien*. SCSS ist eine Stylesheet-Sprache, die die Funktionalitäten von CSS erweitert [Sas21].

Angular bietet zusätzliche Funktionen, die den Entwickler bei der Implementierung von Webanwendung unterstützen. Einige dieser Funktionen wurden in der Arbeit verwendet und werden im Folgenden genauer erläutert.

3.1. Strukturdirektiven

Strukturdirektiven erweitern die Funktionalität von HTML-Elementen. Sie werden im Template verwendet und sind durch einen voranstehenden Stern * markiert. Die ***ngIf-Direktive** ist ein Vertreter der Strukturdirektiven. Die Direktive wird im HTML-Tag angegeben und somit diesem HTML-Element zugeordnet. Dieses HTML-Element kann

durch die Direktive ein- bzw. ausgeblendet werden. Dafür muss der Direktive ein `boolean` zugewiesen werden, dessen Wahrheitswert darüber bestimmt [MHK20][Ang21b]. Weitere Strukturdirektiven sind unter anderem `*ngFor` und `*ngSwitchCase`. Dabei erlaubt die Strukturdirektive `*ngFor` ein wiederholtes Einfügen eines HTML-Elementes in den HTML-Code, während die Direktive `*ngSwitchCase`, ähnlich wie `*ngIf`, eine Alternative formuliert.

3.2. Interpolation

Durch *Interpolation* können Daten (Werte) aus der TypeScript-Klasse ins Template eingebunden werden. Dies geschieht syntaktisch durch zwei geschweifte Klammern. Die Klammern umschließen die Variable aus der TypeScript-Klasse, siehe Listing 3.1. Dadurch wird der Wert der Variablen in der Webanwendung angezeigt [MHK20][Ang21d].

```
1 <p>{{Variable}}</p>
```

Listing 3.1: Interpolation im Template

3.3. Property Binding

Mithilfe von *Property Bindings* können variable Werte aus der TypeScript-Klasse als HTML-Attribut einem HTML-Element zugeordnet werden. Diese Eigenschaft kann z.B. dazu verwendet werden, das `href`-Attribut eines Links zu ändern [MHK20][Ang21c]. Syntaktisch wird bei einem Property Binding die entsprechende Eigenschaft (Property) von eckigen Klammern umschlossen und mit einem Wert in Hochkomma durch das Gleichheitszeichen verknüpft, siehe Listing 3.2.

```
1 <a [href]="Variable"> Link </a>
```

Listing 3.2: Property Binding

3.4. Event Binding

Mit *Event Bindings* kann auf Ereignisse im Template reagiert werden. Mögliche Ereignisse sind das Betätigen (engl. `click`) eines Knopfes (engl. `Button`) oder das Betätigen einer bestimmten Eingabetaste (engl. `Key`). Diese Ereignisse können mit einer Funktion in der

TypeScript-Klasse verknüpft werden. Somit stellen Event Bindings den Datenfluss vom Template zur TypeScript-Klasse dar. Sie sind somit der Gegenpart von Property Bindings. Im Listing 3.3 wird gezeigt, wie beim Betätigen des Buttons die `clickFunktion()` in der TypeScript-Klasse aufgerufen wird [MHK20][Ang21a].

```
1 <button (click)="clickFunktion()"> Click me </button>
```

Listing 3.3: Event Binding

3.5. Backend Anbindung

Die Anbindung an das Backend wird in Angular typischerweise in einem *Service* implementiert. Ein *Service* in Angular ist eine TypeScript-Klasse, die einem konkreten Zweck dient. Ein Service sollte möglichst genau eine Sache erledigen. Ein Service kann typischerweise von allen/vielen Komponenten verwendet werden. In einen solchen Service, der die Anbindung an das Backend implementiert, muss in Angular der `HttpClient`-Service per *dependency injection* injiziert werden. Dieser Service wird durch das Modul `HttpClientModule` bereitgestellt, welches in die `app.module.ts` importiert werden muss. Der `HttpClient`-Service stellt Funktionen `get()`, `put()`, `post()` und `delete()` in Äquivalenz zu den entsprechenden HTTP-Anfragen (bzw. den REST-Endpunkten) bereit.

4. Node.js

5. Datenbanken

Um Daten zu speichern, gibt es verschiedene Lösungsansätze. Generell unterscheidet man zwischen den SQL- und NoSQL-Datenbankansätzen. Die Abkürzung „SQL“ steht dabei für die „Sequel Query Language“, die zur Abfrage von relationalen Datenbanken genutzt wird. Im Gegensatz dazu umfasst der Begriff „NoSQL“ – ausgeschrieben „Not only SQL“ – jene Datenbanksysteme, die nicht den relationalen Ansätzen entsprechen.

5.1. SQL

Spricht man von SQL-Datenbankansätzen, ist die Rede von Relationalen-Datenbank-Management-Systemen (RDBMS). Innerhalb eines RDBMS werden die Daten strukturiert in Tabellen gespeichert, wobei jede Spalte über einen spezifischen Typ verfügt. Systeme relationaler Datenbanken basieren auf dem Konzept sogenannter ACID-Transaktionen.

5.1.1. ACID

ACID ist die Abkürzung vierer Worte, die den Aufbau und die Funktionsweise der RDBMS beschreiben: atomar, konsistent, isoliert und dauerhaft (eng.: Atomic, Consistent, Isolated, Durable). Unter atomar versteht man die Aussage, dass entweder alle Änderungen, die während einer Transaktion vorgenommen werden, übernommen werden oder keine dieser Änderungen. Nachdem die Transaktion durchgeführt wurde, befinden sich die Daten in einem konsistenten Zustand. Damit erhalten alle Abfragen, die bestimmte Daten abrufen möchten, dasselbe aktualisierte Ergebnis. Die Transaktionen, die an einem Datensatz vorgenommen werden, sind außerdem isoliert und damit unabhängig von weiteren Transaktionen. Zuletzt sind Änderungen, die innerhalb des Systems passieren, permanent und gehen im Falle eines Versagens desjenigen Systems nicht verloren.

5.2. NoSQL

NoSQL-Datenbankansätze basieren nicht auf dem Konzept der ACID-Transaktionen. Ihre Charakteristik liegt in der Speicherung von semi- bis hin zu unstrukturierten Daten. Dadurch verfügen sie über ein flexibles Datenmodell und werden gerne für große Datenmengen

im Internet verwendet. Anstelle des ACID-Konzepts liegt die Grundlage der nichtrelationalen Datenbanksysteme in dem BASE-Konzept. Um dieses jedoch verstehen zu können, muss zuerst ein Überblick über das CAP-Theorem verschafft werden. Das Kürzel CAP steht für die Bedingungen Konsistenz, Verfügbarkeit und Teilungstoleranz (eng: Consistency, Availability, Partition Tolerance). Die Konsistenz bedeutet auch hier den einheitlichen Zustand der Daten nach Durchführung einer Transaktion. Unter der Verfügbarkeit versteht man die durchgängige Erreichbarkeit des Systems. Die Teilungstoleranz sagt aus, dass das System auch weiterfunktionieren kann, sollte es auf Gruppen von Servern aufgeteilt werden, die nicht in der Lage sind, miteinander zu kommunizieren. Laut dem Theorem können jedoch immer nur zwei der drei Bedingungen gleichzeitig erfüllt werden. Auf dieser Theorie basiert BASE. Darunter versteht man die grundsätzliche Verfügbarkeit, den weichen Zustand und die letztendliche Konsistenz (eng: Basically Available, Soft state, Eventual consistency). Auf Grundlage des BASE-Konzepts ist das System im Sinne des CAP-Theorems verfügbar. Außerdem verändert sich der Zustand des Systems mit der Zeit, auch wenn dieses keine Eingaben erhält. Dies steht jedoch nicht im Gegensatz zur letztlichen Konsistenz. Auf lange Sicht wird das System einen konsistenten Zustand erreichen, vorausgesetzt, in dieser Zeit werden keine Daten an das System gesendet.

5.3. SQL vs NoSQL

Einen großen Unterschied der beiden Ansätze sieht man in der Flexibilität der Schemata. RDBMS weisen nur eine geringe Flexibilität in ihrem Design auf. Sie sind zeilenorientiert, sodass schon das Hinzufügen einer Spalte zu einer Tabelle zu Problemen führen kann, vor allem, wenn die Tabelle bereits mit Daten gefüllt sein sollte. Oft kann ein solches Hinzufügen dazu führen, dass neue Tabellen erstellt werden. Die Komplexität des Systems wird dadurch erhöht, da es Beziehungen zwischen den Tabellen geben kann. Spaltenorientierte Datenbanken haben den Vorteil, dass es möglich ist – falls nötig – Spalten ohne Probleme hinzuzufügen. Auch Dokumente mit semistrukturierten Daten schaffen keine größeren Komplexitäten und sind in der Wahl des Schemas sehr flexibel. (S.19) Dadurch ist die Arbeit mit allen möglichen Datentypen möglich. Änderungen, die eine Aktualisierung des Datenbankschemas bedeuten, können somit ohne weiter Probleme umgesetzt werden. (S.18) Einen weiteren Vergleichspunkt findet man in der Verwendung komplexer Abfragen. Durch den standardisierten Aufbau von RDBMS entstehen komplexe JOIN-Abfragen, die sowohl schwer zu implementieren und zu pflegen sind als auch erhebliche Ressourcen für die Ausführung benötigen. NoSQL-Datenbanken verfügen weder über Beziehungen noch Fremdschlüssel, wodurch keine komplexen Abfragen entstehen. Jedoch müssen mehrere Abfragen ausgeführt werden, um über eine Tabelle hinaus abfragen zu können. (S.19/20) Die einfachen Datenmodelle führen dennoch zu einer einfachen Verwaltbarkeit. (S.18) Ebenfalls

wichtig zu betrachten ist die Skalierbarkeit. Hierbei gilt es, die beiden Ansätze von Scale-up und Scale-out zu erläutern. SQL-Datenbanken nutzen den Scale-up-Ansatz. Dabei erfolgt die Erweiterung vertikal mithilfe von kostspieligen, qualitätsvollen Servern. Dieser Ansatz versagt jedoch, sobald die Transaktionsraten und Anforderungen an schnelle Reaktionen steigen. In Momenten wie diesen kommt der Scale-out-Ansatz zum Einsatz. Dabei findet eine horizontale Erweiterung unter Verwendung von günstigen Commodity-Servern statt. (S.18) Diesen Ansatz nutzen NoSQL-Datenbanken, wodurch sie eine bessere Skalierbarkeit bieten, als auch Daten preiswert speichern und verarbeiten können. (S.19/20) Ein letzter wichtiger Vergleichspunkt ist die Synchronisation der Daten. Das Aktualisieren der Daten über die Tabellen der RDBMS hinweg erweist sich als sehr komplex. Die Updates müssen dabei auf die verschiedenen Knoten des Systems verteilt werden. Unterstützen Systeme ein gleichzeitiges Schreiben auf verschiedenen Knoten nicht, kann es dabei zu Fehlern und zu höheren Latenzzeiten kommen. Um diese Latenzzeiten gering zu halten, bieten dagegen NoSQL-Anwendungen gute Synchronisierungsmöglichkeiten. Beispielsweise MongoDB kann eine Aktualisierung gleichzeitig auf mehreren Knoten ausführen und die Konsistenz innerhalb des Systems in einer akzeptablen Latenzzeit sicherstellen. (S.19/20) Für die Arbeit wird mit der Verwendung der Datenbank MongoDB ein NoSQL-Ansatz gewählt, da die Vorteile vor allem in Bezug auf die Flexibilität der Schemata überwiegen. Ebenso andere Begründungen zur Nutzung von NoSQL-Ansätzen, die in diesem Kapitel beschrieben werden, sind für die Arbeit von Vorteil.

6. MongoDB

6.1. Überblick

MongoDB ist eine NoSQL-Datenbank, die über einen JSON-basierten Dokumentenspeicher verfügt. Abfragen in MongoDB basieren auf den Schlüsseln innerhalb der Dokumente. Jene Dokumente können auf verschiedenen Servern verteilt sein. Führt man eine Abfrage aus, sucht in seinen Dokumenten nach Resultaten, die er als Ergebnis ausgibt. Dadurch erreicht man eine lineare Skalierbarkeit und eine verbesserte Performance. Außerdem verfügt MongoDB über eine Primär-Sekundär-Replikation, wobei die Primärseite die Schreibanforderungen annimmt. Will man die Schreibleistung verbessern, kann man das sogenannte Sharding einsetzen. Dabei werden die Daten auf mehrere Rechner verteilt, die jeweils unterschiedliche Teile der Datensätze aktualisieren können. In MongoDB wird Sharding automatisch ausgeführt. Fügt man einen neuen Rechner hinzu, werden die Daten automatisch verteilt. (S.26)

6.2. Speicherung der Daten

6.2.1. JSON-basierte Speicherung

Eine Datenbank enthält Kollektionen, die Dokumente umfassen. Diese Dokumente enthalten die zu speichernden Daten (S.30) in Form von Schlüssel-Wert-Paaren (S.27). Zur Speicherung wird Binary JSON – kurz BSON – eingesetzt. JSON ist die JavaScript Object Notation und gilt als einer der Standardsprachen für den Datenaustausch im heutigen, modernen Web. Das Format von JSON ist sowohl für Maschinen als auch für den Menschen lesbar. Zudem werden alle Standarddatentypen wie beispielsweise Strings, Zahlenwerte, Boolean-Werte und Arrays unterstützt und können verwendet werden. (S.31) Ein Vorteil von JSON ist die Möglichkeit, zusammengehörige Daten gruppiert an einem Platz zu speichern. Dadurch sind gewisse Daten einfacher auffindbar, was die Performance der Abfragen verbessert. Außerdem ist es möglich, schemalose Modelle zu verwenden, wodurch Änderungen an einem Schema nie zu Problemen führen können. Im Gegenteil zu RDBMS kann für eine Speicherung ein dynamisches Schema gewählt werden. Dies bedeutet, dass Dokumente über unterschiedliche Strukturen oder Felder verfügen oder aber auch ähnliche

Felder unterschiedliche Daten speichern können. (S.26/27) In Fällen, in denen eine Kollektion Dokumente unterschiedlicher Typen beinhaltet, spricht man von einem polymorphen Schema. (S.32)

6.2.2. BSON

Ein JSON-Dokument enthält die tatsächlichen Daten. Dennoch werden jene Daten nicht in dieser Form abgespeichert, sondern zuvor binär verschlüsselt. Die Implementierung eines BSON-Dokuments in MongoDB ist schnell und leichtgewichtig. Außerdem ermöglicht es MongoDB, Indizes in Objekten zu erstellen und diese mit den Abfragen abzulegen. (S.31)

6.2.3. Identifier

Wie auch die Primärschlüssel in RDBMS werden Identifier in Dokumenten benötigt, um diese identifizieren zu können. Identifier werden mit dem Schlüssel `id` bezeichnet. Den Wert der ID kann man – sofern es gewollt ist – selbst zuordnen. Dafür kann jeder beliebige Datentyp verwendet werden, außer Arrays. Ordnet man selbst der ID keinen Wert zu, übernimmt dies MongoDB automatisch.

7. Progressive Web App

Wie in Kapitel 1.2 erwähnt verfügt eine PWA unter anderem über folgende Funktionen:

- Installierbar,
- Zugriff auf Geräteschnittstellen,
- Netzwerkunabhängig,
- Push-Notifikations.

Im folgenden werden die theoretischen Grundlagen erläutert, die benötigt werden, um diese Funktionalitäten zu realisieren.

7.1. Web-App-Manifest

The web app manifest is a JSON file that defines how the PWA should be treated as an installed application, including the look and feel and basic behavior within the operating system [Dev22].

Eine PWA kann auf ein Endgerät wie zum Beispiel ein Desktop oder Handy installiert werden. Um diese Funktion zu realisieren, müssen zusätzliche Informationen wie zum Beispiel der Name und das Icon der installierten Applikation in einer Datei festgehalten werden.

Bei dieser Datei handelt es sich um das sogenannte Web-App-Manifest. Die Informationen sind im JavaScript Object Notation ([JSON](#))-Format¹ angegeben.

Ohne Das Manifest ist die Applikation nicht installierbar, somit ist die Datei eine zwingende Voraussetzung für eine PWA. Das Manifest muss mindestens ein `name`-Schlüssel und ein `String`-Wert aufweisen. Neben dem Namen der Applikation kann ein Manifest über folgende Informationen verfügen:

¹ durch Komma getrennte Schlüssel-Wert paare

7.1.1. short_name

Unter `short_name` kann ein kurzer Name der Applikation angegeben werden. Dieser Name wird verwendet, falls das Endgerät nicht über genügend Platz verfügt, um den Originalen Namen anzuzeigen.

7.1.2. icons

Unter `icons` wird ein Array¹ mit Bildobjekten gespeichert. Ein Bildobjekt besteht aus einem Dateipfad, unter dem das anzuzeigende Bild gespeichert ist, einer Typ Beschreibung des Bildes zum Beispiel `png` oder `svg`, eine Informationen über die Auflösung des Bildes und optional noch eine Angabe welchem Zweck das Bild dient.

Die Gespeicherten Bilder werden als App-Icon auf dem Desktop oder Handy angezeigt.

7.1.3. start_url

Die angegebene `start_url` ist jene [URL](#) die geöffnet wird, sobald der Nutzer das installierte Icon auswählt und somit die Applikation startet. Wird keine explizite Startadresse angegeben, so wird die URL verwendet, von der die PWA installiert wurde.

7.1.4. display

Beim Auswählen des Installierten Icons wird die PWA in einem neuem Fenster geöffnet. Unter `display` kann angegeben werden, wie das Betriebssystem das Fenster darstellen soll. Es kann zwischen `Fullscreen`, `Standalone` und `Minimal User Interface` unterschieden werden. Der Unterschied zwischen den einzelnen Auswahlmöglichkeiten liegt bei den Navigationselementen, siehe Abbildung 7.1 und 7.2.



Abbildung 7.1.: display in Standalone Einstellung

¹ Datentyp, das mehrere Werte speichern kann



Abbildung 7.2.: display in Minimal User Interface Einstellung

7.1.5. theme_color

Mit Hilfe dieser Einstellung kann die Farbe der oberen Navigationsleiste angepasst werden, wie in Abbildung 7.3 dargestellt ist. Hierbei ist jedoch darauf zu achten, das die Applikation nicht den meta tag theme-color definiert.



Abbildung 7.3.: Theme Color auf Weiß geändert

7.1.6. Debugging vom Manifest

Neben den oben aufgezählten Grundeinstellungen sind viele weitere Möglich. Um nachzuvollziehen, ob alle Einstellungen den Anforderungen entsprechen kann das Manifest mithilfe der Browser Entwicklerwerkzeuge untersucht werden. Unter Google Chrome kann unterm Reiter *Application* das Manifest ausgewählt werden. Darauf hin erhält man folgende Ansicht, siehe Abbildung 7.4.

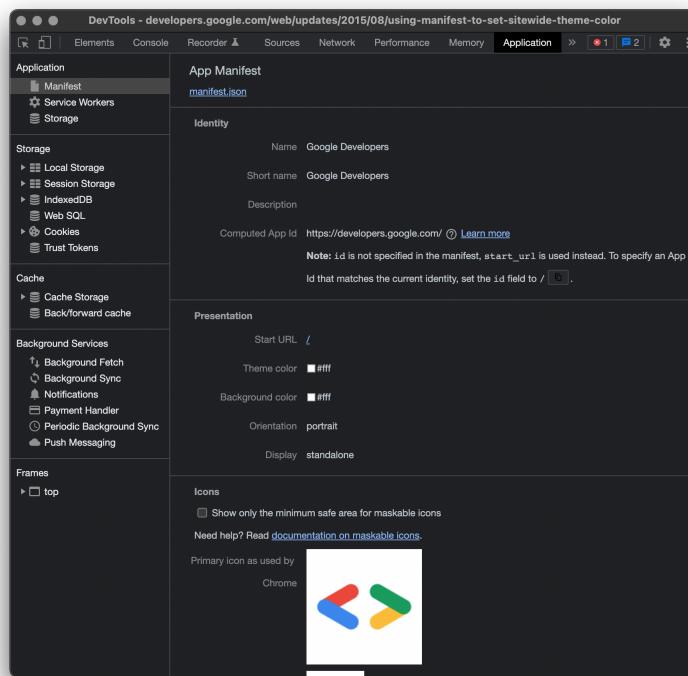


Abbildung 7.4.: Entwicklereinstellungen Web-App-Manifest

7.2. Service Workers

Service workers are a fundamental part of a PWA. They enable fast loading (regardless of the network), offline access, push notifications, and other capabilities [Dev].

Der *Service Worker* ist ein wichtiger Grundbaustein um Funktionalitäten wie Push-Notifikations, Hintergrund-Synchronisation und die Möglichkeit, auch Offline die Anwendung auszuführen, zu realisieren.

Bei dem *Service Worker* handelt es sich um ein script das im Hintergrund des Browsers, unabhängig von der Webanwendung, läuft [Gau]. Entstanden sind die service worker aus der Verwendung des Application Caches . Die service Worker Applikation Programming Interface (API) wächst kontinuierlich und bietet zunehmende weitere Funktionalitäten.

Bei der Verwendung eines Service Worker sollten folgenden Eigenschaften berücksichtigt werden:

- Ein service worker kann zwar nicht direkt das Document Object Model ([DOM](#)) einer Seite manipulieren, kann aber auf Requests der Seite mit Responses reagieren und die Seite selbst kann darufhin ihr DOM ändern
- Ein service worker ist ein „programmierbarer“ Proxy, der steuert, wie Requests von der Webseite behandelt werden.
- Service workers verwenden die IndexDB API, um client-seitig strukturierte Daten persistent zu speichern.
- Service workers verwenden Promises.

7.2.1. Der Lebenszyklus eines Service Workers

Der Service Worker wird vom Browser in einem eigenen Thread unabhängig der Applikation ausgeführt. Dementsprechend besitzt der Service Worker einen eigenen Lebenszyklus, der im folgenden genauer beschrieben wird.

Registrieren

Der Lebenszyklus eines Service Workers beginnt mit der Registrierung. Dies erfolgt mithilfe der `register()` Methode in der `serviceWorker` Eigenschaft des `navigator`-Objektes [doc], siehe Listing 7.1 Zeile 2. Um das Konzept von Progressive Enhancement zu folgen (Abschitt 1.2), sollte vor der Registrierung überprüft werden, ob die Service-Worker-Schnittstellen vorhanden sind. Hierfür wird die `serviceWorker` Eigenschaft im `navigator`-Objekt abgefragt, siehe Listing 7.1 Zeile 1.

```
1 if ('serviceWorker' in navigator){  
2   navigator.serviceWorker.register('./sw.js')  
3     .then(registration => console.log(registration))  
4     .catch(error => console.error(error));  
5 }
```

Listing 7.1: Registrierung des Service Workers

Das Aufrufen der Methode `register()` gibt ein Promise zurück, welches in Zeile 3 und 4 behandelt wird. Im Erfolgsfall (`then`-Methode) wird ein Objekt von typ `ServiceWorkerRegistration` übergeben. Das Objekt verfügt unter anderem über folgende Schnittstellen:

- mit der `update()` Methode wird die Aktualisierung des Service Workers ausgelöst,
- die Methode `unregister()` hebt die Registrierung des Service Workers auf, welches im anschluss vom Browser gelöscht wird,
- die Eingenschaft `updatefound` wird aufgerufen, sobald der Browser eine neue Version des Service Workers gefunden hat.

Installieren

Mit der Registrierung erfolgt auch die Installation des Service-Worker-Skriptes, welches bei der Registrierung übergeben wurde, siehe Listing 7.1 Zeile 2. In der Installationsphase läd der Service Worker Ressourcen vom Webserver und speichert diese Lokal ab. Wichtig ist hierbei das die Installationsphase erst dann beendet werden darf, wenn alle benötigten Ressourcen heruntergeladen wurden. Um dies zu realisieren steht die Methode `waitFor()` zur Verfügung, die eine Promise-Kette abarbeitet. Die Methode öffnet den zugehörigen Cache. Im Cache werden durch die `addAll()`-Methode alle benötigten Ressourcen gespeichert, die normalerweise beim öffnen der Applikation von Webserver angefordert werden. Dies ermöglicht, das die Applikation in Zukunft auch ohne Netzwerkverbindung aus dem lokalen Cache geladen werden kann.

In Angular wird in der `ngsw-config.json`-Datei definiert, welche Ressourcen in den Cache gespeichert werden sollen. Die Grundeinstellung sieht folgendermaßen aus, siehe Listing 7.2. Aus der JSON-Datei wird ersichtlich, dass sämtliche JavaScript und CSS Dateien sowie die `index.html` im Cache gespeichert wird. Zusätzlich werden Dateien, wie zum Beispiel Bilder, aus dem `assets` Verzeichnis gespeichert.

Listing 7.2: Angular *ngrx-store*-Datei zu Angabe der Ressourcen, die durch den Service Worker in den Cache gespeichert werden sollen.

Die im Cache gespeicherten Dateien können mithilfe der Google Chrome Developer Tools unter dem Reiter *Application* eingesehen werden, siehe Abbildung 7.5. Da es sich in der Abbildung um eine Angular-Anwendung handelt, können die in der *ngsw-config.json*-Datei angegebene Ressourcen wiedergefunden werden.

Aktivieren

Nach der Installation geht der Service Worker in den aktiven Zustand über. Ein Aktivierter Service Worker ist in der Lage funktionale Ereignisse zu behandeln wie zum Beispiel das beantworten von HTTP(S) Anfragen oder das Entgegennehmen von Pushbenachrichtigungen. Der Service Worker übernimmt somit die Kontrolle der Applikation. Dies geschieht jedoch erst wenn ein neuer Browser-Kontext geöffnet wird. Würde der Service Worker während der Laufzeit der Webanwendung aktiviert, kontrolliert er alle HTTP(S)-Anfragen

der Webanwendung, darunter auch jene deren Ressourcen noch nicht im Cache gespeichert wurden.

Überschreiben

Sobald ein neuer Service Worker für einen bestimmten Scope Aktiviert wurde, terminiert der Webbrower das alte Skript. Der alte Service Worker geht in den zustand *redundant* und wird im weiteren Verlauf entfernt.

Unter Apple Safari wird ein registrierter Service Worker automatisch deinstalliert, sobald die Webseite des Service Workers mehrere Wochen nicht aufgerufen wurde.

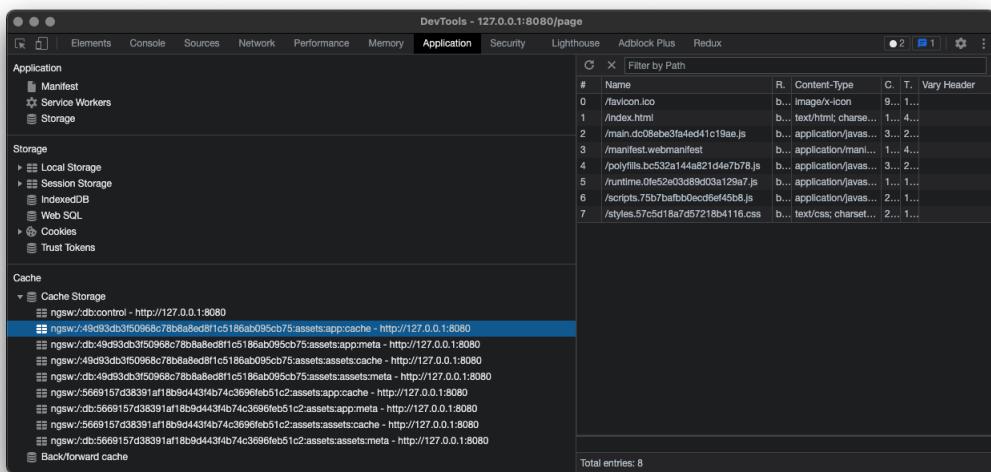


Abbildung 7.5.: Google Chrome Developer Tools zum einsehen der im Cache gespeicherten Dateien.

7. Progressive Web App

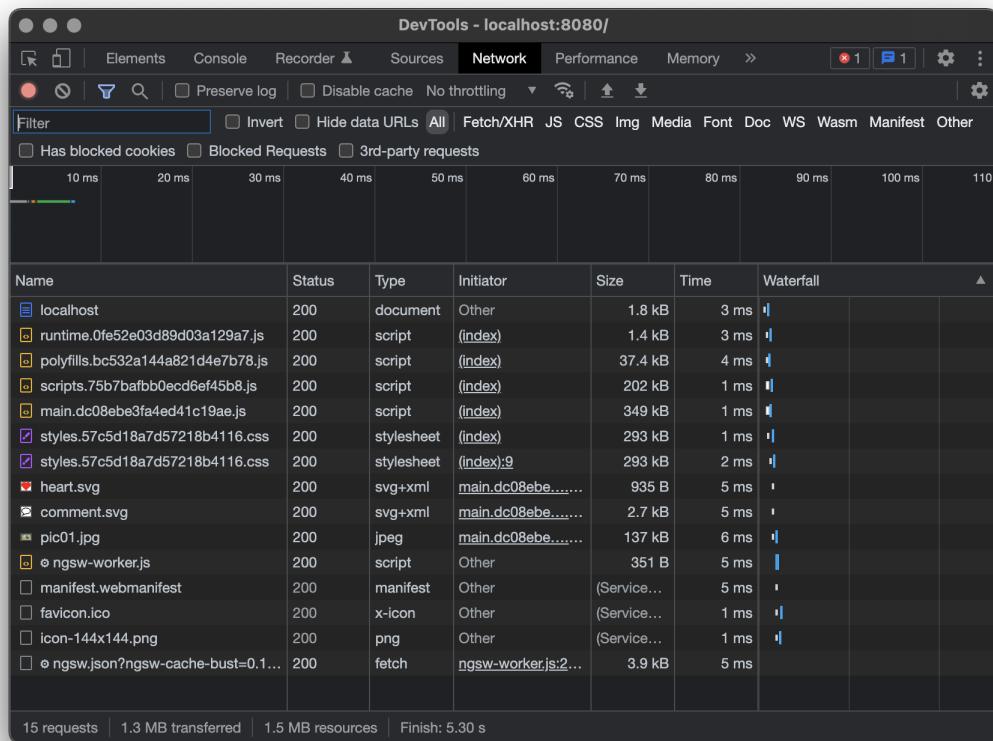


Abbildung 7.6.: Laden einer Applikation ohne Service Worker

7. Progressive Web App

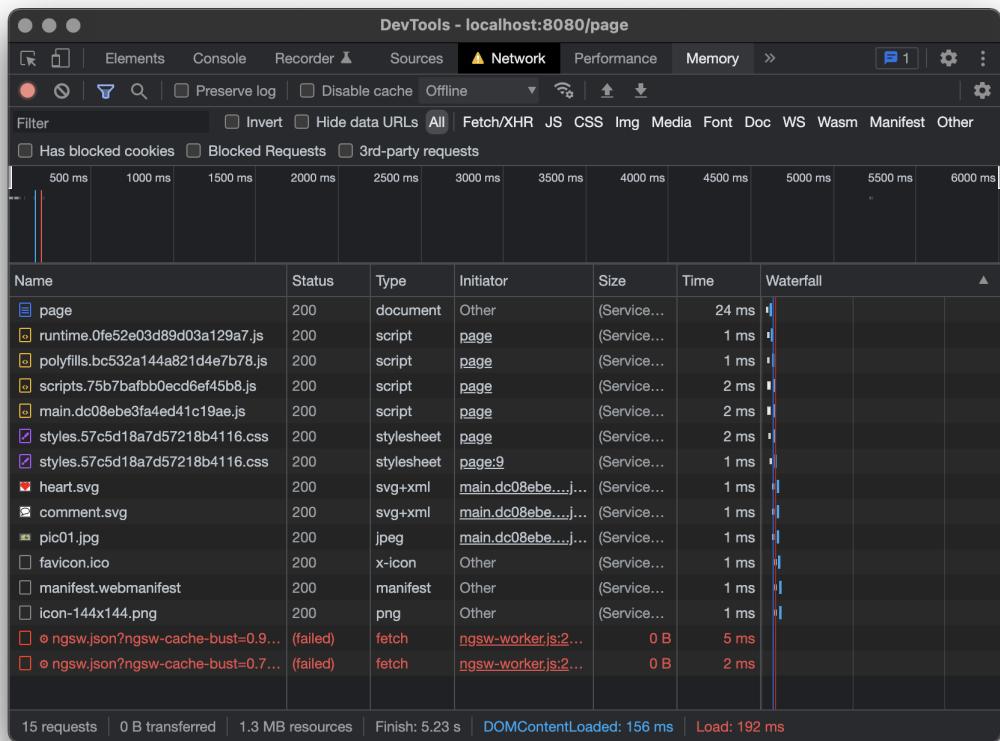


Abbildung 7.7.: Laden einer Applikation ohne Internetverbindung

7.3. Push-Notifikations

Eine Notifikation ist eine Nachricht, die auf dem Endgerät des Nutzers angezeigt wird. Dies kann als Reaktion auf eine Nutzereingabe geschehen, es kann jedoch auch unabhängig vom Nutzer eine Nachricht von einem Server „gepusht“ werden und das auch ohne das die Applikation aktiv ist. Um push Notifikations zu erzeugen werden zwei APIs benötigt. Die *Notifikations API* visualisiert die Nachricht für den Nutzer und die *Push API* erlaubt es den Service Worker, Kapitel 7.2, Nachrichten zu verwalten, die vom Server gesendet wurden, während die Applikation nicht aktiv war.

Neben dem Frontend, Backend und Service Worker sind an der Push-Kommunikation noch der Push-Service und Push-Dienst beteiligt. Jeder Browserhersteller hat einen eigenen Push-Dienst (eng. *unser agent*) implementiert. Google Chrome verwendet *Firebase Cloud Messaging*, Mozilla Firefox den *Mozilla Push Service* und Microsoft Edge verwendet die *Windows Push Notification Services*. Der Push-Dienst nimmt die Pushnachrichten vom Push-Service entgegen, und leitet diese an den Service Worker weiter. Die Kommunikation mit den Push-Diensten ist durch die Internet Engineering Task Force ([IETF](#)) spezifiziert [[TDR16](#)]. Somit müssen keine gesonderten Implementierung für die Einzelnen Push-Dienste vorgenommen werden.

Wie genau die einzelnen Kommunikationspartner miteinander interagieren ist in dem Sequenzdiagramm in Abbildung 7.8 dargestellt, und wird im folgendem genauer erläutert.

7.3.1. Push-Registrierung

Um Push-Nachrichten empfangen zu können, muss die Webanwendung über den Webbrowsr eine Push-Registrierung beim Push-Dienst beantragen. Die Beantragung erfolgt über die PushAPI die über den ServiceWorker, genauer den `ServiceWorkerRegistration` zur Verfügung gestellt wird. Aus diesem Grund ist es zwingend notwendig, dass der Service Worker registriert ist. Bei der Registrierung sollte das Konzept von *Progressive Enhancement* berücksichtigt werden, indem die Browserkompatibilität vor der Registrierung abgefragt wird, siehe Listing 7.3. Diese Abfrage ist wichtig, da der Apple Safari Browser keine Push-Funktionen unterstützt, siehe Abbildung 7.9 (b).

Falls der Browser über die nötigen Funktionen verfügt, kann die `subscribe()`-Methode von `ServiceWorkerRegistration` aufgerufen werden, welches eine Objekt vom Typ `PushSubscription` zurück gibt. Die `PushSubscription` verfügt über ein Konfigurationsobjekt mit zwei Eigenschaften. Die erste Eigenschaft mit dem Namen `userVisibleOnly`, ist eine boolsche Eigenschaft, die beschreibt, ob Pushnachrichten immer zu einer für den Anwender sichtbaren Meldung führt. Diese Eigenschaft hat standardmäßig den Wert `false`.

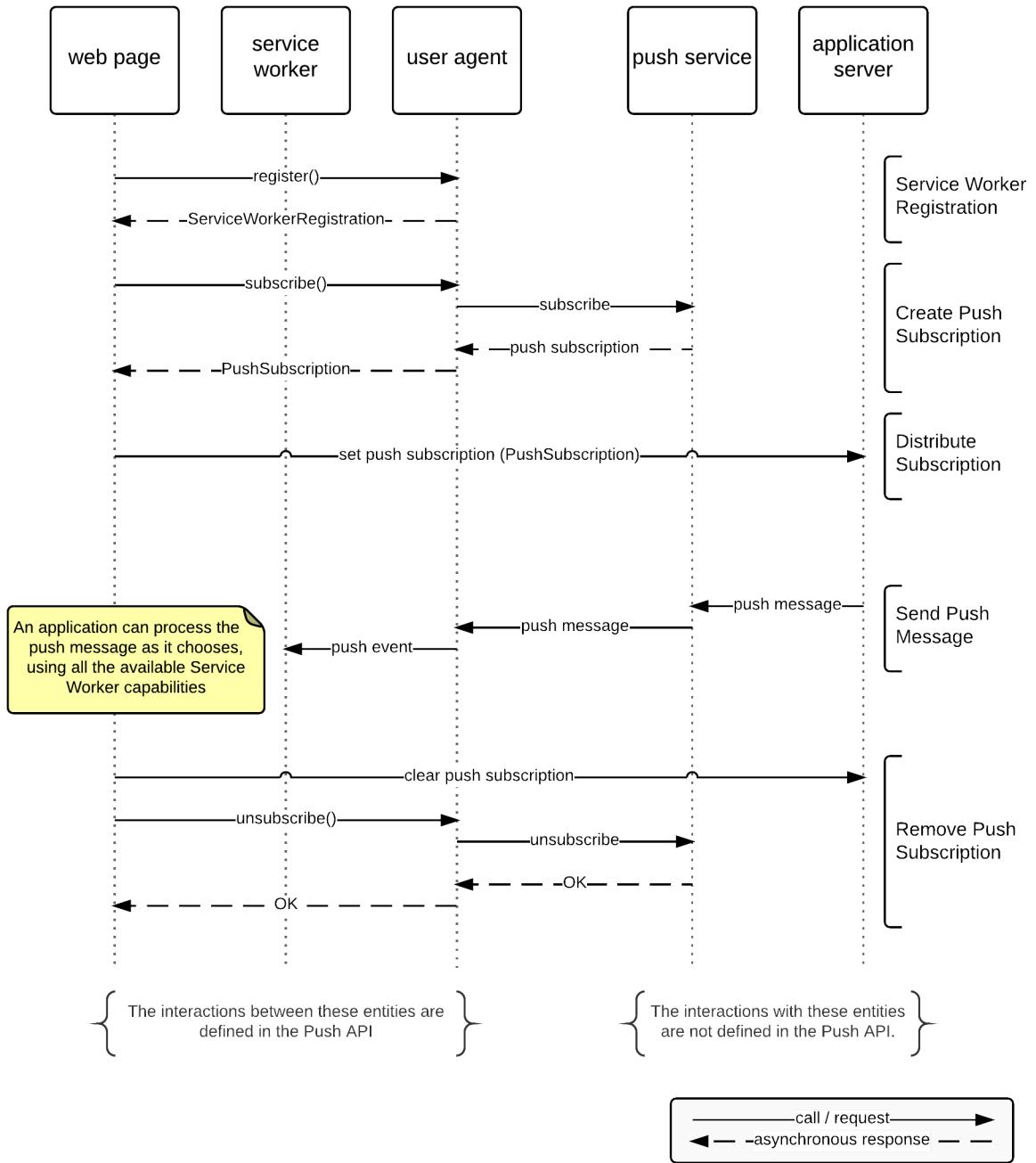


Abbildung 7.8.: Sequenzdiagramm für die Ereignisse einer Anmeldung, push Nachrichtentransfer und Abmeldung [BT16]

Dies führt aber bei dem Browser Google Chrome zu Komplikationen, da der Browser aus Gründen der Datensicherheit keine sogenannte *Silent Push* Nachrichten zulässt.

Die zweite Eigenschaft des Konfigurationsobjektes hat den Namen `applicationServerKey`. Diese Eigenschaft wird benötigt um die Nachrichten zwischen Push-Dienst und Anwendung zu verschlüsseln. Zur Verschlüsselung wird das Voluntary Application Server Identification (**VAPID**) Protokoll verwendet, das von IETF spezifiziert wurde [TB17]. Die Verschlüsselung

benötigt einen öffentlichen und einen privaten Schlüssel. Der öffentliche Schlüssel ist unter der `applicationServerKey` Eigenschaft gespeichert. Der private Schlüssel wird wiederum im Backend der Anwendung hinterlegt.

```

1  if('serviceWorker' in navigator){
2      navigator.serviceWorker.register('sw.js');
3      navigator.serviceWorker.ready.then(registration => {
4          if('PushManager' in window){
5              registerForPush(registration.pushManager);
6          }
7      });
8  }

```

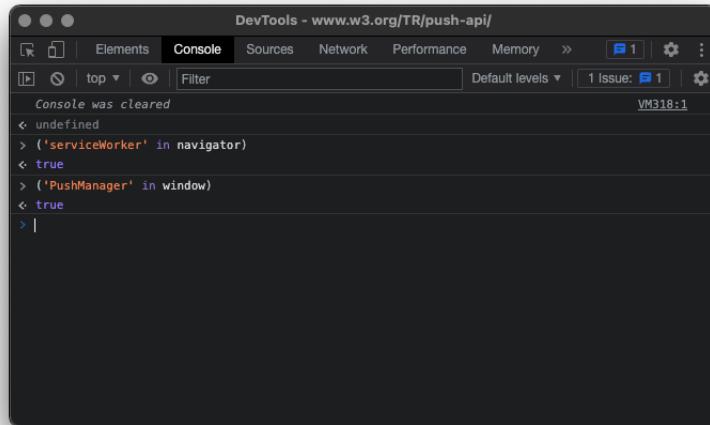
Listing 7.3: Push-Registrierung unter Berücksichtigung der Konzepte von Progressive Enhancement

7.4. Progressive Web Apps in Angular

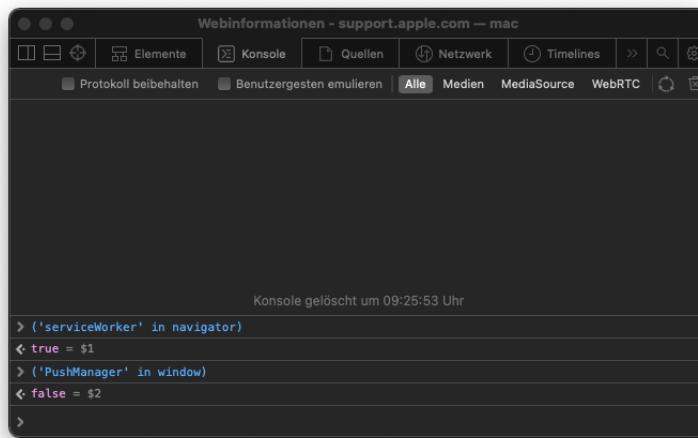
Angular (Abschnitt 3) eignet sich sehr gut um eine PWA zu entwickeln, da beides Entwicklungen von Google sind. Um aus einer Angular Anwendung eine PWA zu erstellen, muss das Paket `@angular/pwa` über das Command Line Interface ([CLI](#)) hinzugefügt werden. Durch das hinzufügen des Paketes werden im Hintergrund folgende Veränderungen an dem Projekt durchgeführt:

- Das Paket `@angular/service-worker` wird zum Projekt hinzugefügt,
- der Build Support für den Service Worker wird in der Angular CLI aktiviert,
- das `ServiceWorkerModule` wird im `AppModule` importiert,
- die Datei `index.html` wird um ein Link zum Web App Manifest (Abschnitt 7.1) und um relevante Meta-Tags ergänzt,
- Icon-Dateien werden erzeugt und verlinkt,
- die Konfigurationsdatei `ngsw-config.json` für den Service Worker wird erzeugt.

Im anschluss kann die Applikation alle Grundfunktionen einer PWA ausführen. Die Anwendung kann installiert werden. Im Web App Manifest steht der Projektname und das unter `assets` abgespeicherte Angular Icon wird als App-Icon verwendet. Nach dem erstmaligen öffnen der Applikation im Browser wird der Service Worker installiert. Anschließend kann die Webseite auch ohne Netzwerkverbindung geladen werden.



(a)



(b)

Abbildung 7.9.: Überprüfung der Pushnachricht Funktionalitäten in den Browsern (a) Google Chrome , (b) Apple Safari

7.5. PWA für IOS

Apple unterstützt die Entwicklung von PWAs nicht. PWAs stellen eine Möglichkeit dar, um Apps für das iPhone zu installieren, die sich nicht im App Store befinden. Im Jahr 2020 hat Apple mit dem App Store ein Umsatz von 643 Milliarden US-Dollar erwirtschaftet [Kir]. Apple erhält 30% Provision, wenn eine App gekauft oder wenn In-App Käufe abgeschlossen werden. Durch PWAs könnte diese Einnahmequelle wegfallen.

Durch zusätzliche Einstellungen können PWAs auf dem Homescreen von iOS Geräten installiert werden. Im `head` der `index.html` muss der Dateipfad zum verwendeten Icon angegeben werden, da dieser Pfad nicht aus dem Web-App-Manifest ausgelesen wird. Des Weiteren muss in einem `meta-tag` angegeben werden, dass die Webanwendung als App

genutzt werden kann. Diese zusätzlichen Angaben ermöglichen es die Applikation zu installieren, Apple beschränkt jedoch die volle Funktionsumfang der PWAs. Durch ein Softwareupdate hat Apple eine Zwangslösung für lokal beschreibbare Speicherfunktionen eingeführt [t3n]. Eine PWA nutzt diesen lokalen Speicher um Daten zu speichern. Auch der Service Worker wird von Safari nicht vollständig unterstützt. Safari ermöglicht weder Push-Nachrichten noch Hintergrund Synchronisation [med].

7.6. Endgerät Emulation

Um eine PWA Lokal auf Endgeräten zu testen, können diese Emuliert werden. Um ein Android Handy zu Emulieren, kann das Programm *Android Studio* verwendet werden. In *Android Studio* kann mithilfe des *Android Virtual Device Manager* (**AVD Manager**) ein Virtuelles Handy gestartet werden, Abbildung 7.10 a. Im Emulierten Handy kann der Browser geöffnet werden. Um auf die Lokale IP-Adresse des Rechners zugreifen zu können muss im Browser des Handys diese (10.0.2.2:8080) IP-Adresse eingegeben werden, Abbildung 7.10 b. Daraufhin öffnet sich die Anwendung, die unter Einstellung installiert werden kann, Abbildung 7.11 a. Die Anwendung wird darauf hin zum Startbildschirm hinzugefügt, Abbildung 7.11 b.



Abbildung 7.10.: (a) Emuliertes Android Handy in Android Studio, (b) PWA im Browser geöffnet

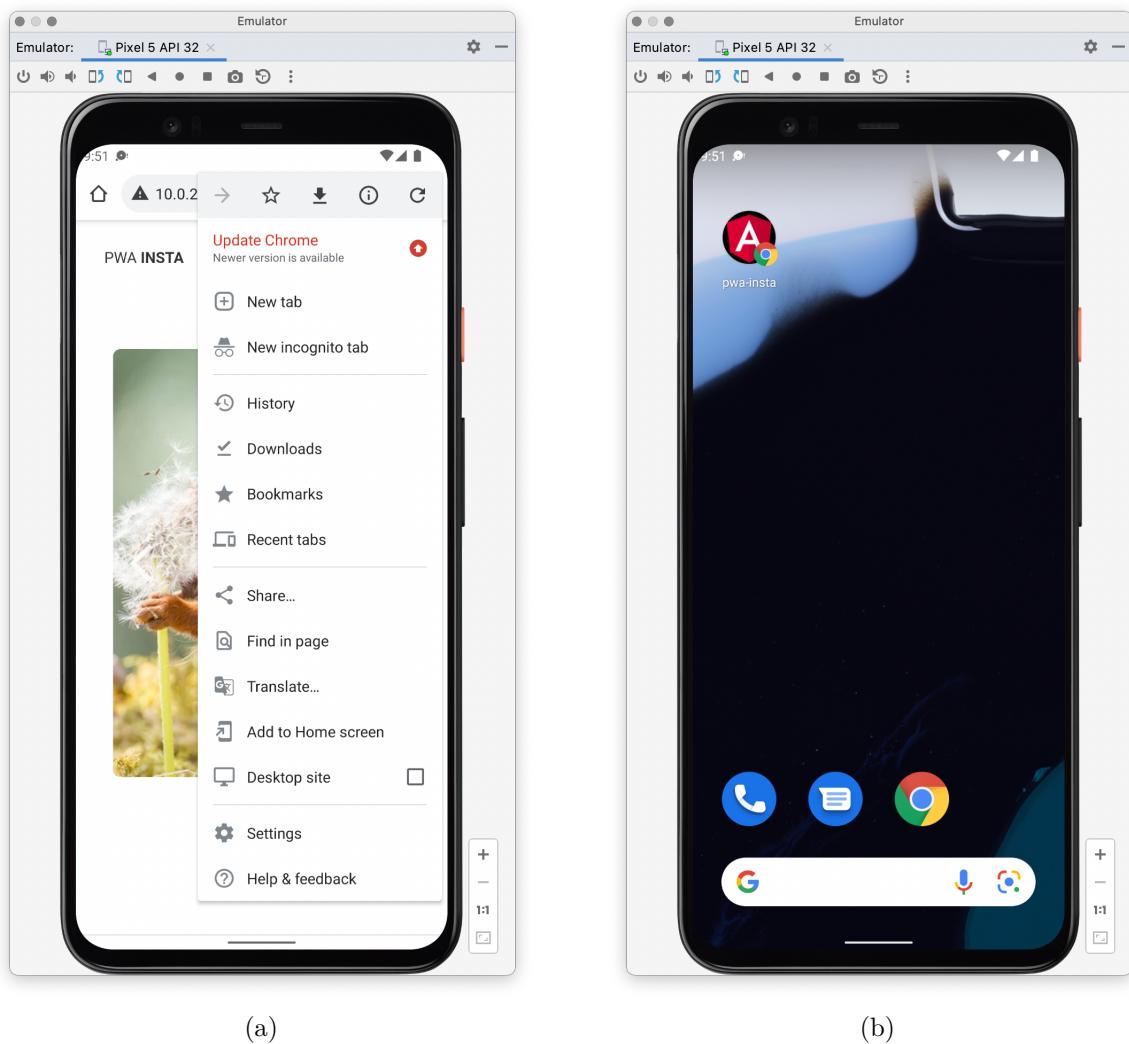


Abbildung 7.11.: (a) Installation der PWA unter Android, (b) Installierte PWA

Literatur

- [Ang21a] Angular. *Event Binding*. 2021. URL: [Event%20binding](#).
- [Ang21b] Angular. *NgIf*. 2021. URL: <https://angular.io/api/common/NgIf>.
- [Ang21c] Angular. *Property binding*. 2021. URL: <https://angular.io/guide/property-binding>.
- [Ang21d] Angular. *Text interpolation*. 2021. URL: <https://angular.io/guide/text-interpolation>.
- [BT16] Peter Beverloo und Martin Thomson. *Push API*. März 2016. URL: <https://www.w3.org/TR/push-api/#abstract>.
- [Cha] Steve Champeon. *PROGRESSIVE ENHANCEMENT AND THE FUTURE OF WEB DESIGN*. URL: http://www.hesketh.com/progressive_enhancement_and_the_future_of_web_design.html.
- [Dev] Google Developers. *Service workers*. URL: <https://web.dev/learn/pwa/service-workers/>.
- [Dev22] Google Developers. *Web app manifest*. 27. Feb. 2022. URL: <https://web.dev/learn/pwa/web-app-manifest/>.
- [doc] mdm web docs. *Navigator*. URL: <https://developer.mozilla.org/en-US/docs/Web/API/Navigator>.
- [Gau] Matt Gaunt. *Service Workers: an Introduction*. URL: <https://developers.google.com/web/fundamentals/primers/service-workers>.
- [Kir] Julius Kirchenbauer. *Apple Entwickler:innen stellen sich den Herausforderungen während der Pandemie und steigern die Verkäufe und Umsätze, die das App Store-Ökosystem ermöglicht hat, um 24 Prozent auf 643 Milliarden US-Dollar im Jahr 2020*. URL: <https://www.apple.com/de/newsroom/2021/06/apple-developers-grow-app-store-ecosystem-billings-and-sales-by-24-percent-in-2020/#:~:text=Apple%20hat%20heute%20bekannt%20gegeben,Prozent%20im%20Vergleich%20zum%20Vorjahr..>
- [Mc21] Mozilla und individual contributors. *Webtechnologien für Entwickler*. 2021. URL: <https://developer.mozilla.org/de/docs/Web>.
- [med] mediaevent. *iOS Progressive Web Apps – PWA – auf iPad und iPhone*. URL: <https://www.mediaevent.de/javascript/ios-pwa.html>.

- [MHK20] F. Malcher, J. Hoppe und D. Koppenhagen. *Angular: Grundlagen, fortgeschrittene Themen und Best Practices – inkl. RxJS, NgRx und PWA*. dpunkt.verlag, 2020. ISBN: 9783969100820. URL: <https://books.google.de/books?id=Oe8BEAAAQBAJ>.
- [Sam] Sam Richard,Pete LePage. *What are Progressive Web Apps?* URL: <https://web.dev/progressive-web-apps/>.
- [Sas21] Sass. *Sass Basics*. 2021. URL: <https://sass-lang.com/guide>.
- [sta21] stackoverflow. *Web frameworks survey*. 2021. URL: <https://insights.stackoverflow.com/survey/2021#section-most-popular-technologies-web-frameworks>.
- [t3n] t3n. *Artikel merkenUnter dem Deckmantel des Guten: Apples neuer Safari-Browser hindert die Entwicklung von progressiven Web-Apps*. URL: <https://t3n.de/news/deckmantel-guten-apples-neuer-1266784/>.
- [TB17] Martin Thomson und Peter Beverloo. *Voluntary Application Server Identification (VAPID) for Web Push*. RFC 8292. Nov. 2017. DOI: [10.17487/RFC8292](https://doi.org/10.17487/RFC8292). URL: <https://www.rfc-editor.org/info/rfc8292>.
- [TDR16] Martin Thomson, Elio Damaggio und Brian Raymor. *Generic Event Delivery Using HTTP Push*. RFC 8030. Dez. 2016. DOI: [10.17487/RFC8030](https://doi.org/10.17487/RFC8030). URL: <https://www.rfc-editor.org/info/rfc8030>.
- [Typ21] TypeScript. *TypeScript for the New Programmer*. 2021. URL: <https://www.typescriptlang.org/docs/handbook/typescript-from-scratch.html>.
- [W3S21a] W3Schools. *CSS Tutorial*. 2021. URL: <https://www.w3schools.com/css/default.asp>.
- [W3S21b] W3Schools. *HTML Tutorial*. 2021. URL: <https://www.w3schools.com/html/default.asp>.
- [W3S21c] W3Schools. *JavaScript Tutorial*. 2021. URL: <https://www.w3schools.com/js/default.asp>.

Anhang

.1. HTML

Mit der [HTML](#) wird das Grundgerüst einer Internetseite aufgebaut. Dafür werden Textelemente von einem HTML-Tag jeweils geöffnet (`<html>`) und geschlossen (`</html>`). Auch HTML-Tags können Einfluss auf die Position und die Darstellung der Textelemente im Browser ausüben. Zum Beispiel verursacht der HTML-Tag `<h1>Text</h1>`, dass das Textelement als Überschrift angezeigt wird [W3S21b].

Eine HTML-Datei verfügt im Allgemeinen über folgendes Grundgerüst.

```
1 <!DOCTYPE html>
2 <html>
3     <head>
4         <title>Seitentitel</title>
5     </head>
6     <body>
7         <h1> Das ist eine Grosse Ueberschrift </h1>
8         <p> Das ist ein Absatz </p>
9     </body>
10 </html>
```

Listing 4: Grundgerüst einer HTML-Seite

Der Tag `<!DOCTYPE html>` gibt dem Browser an, dass es sich um eine HTML-Datei handelt. Der `<head>`-Bereich beinhaltet Metadaten über das Webdokument, wie zum Beispiel den Titel, der im Browser-Reiter angezeigt wird. Neben dem Titel können im Head noch weitere Metadaten wie Schlüsselwörter, Autor und Zeichenkodierung angegeben werden. Wird das Aussehen einer Webseite in einer separaten Datei festgelegt, muss diese Datei auch im Head der HTML-Datei verlinkt werden. Die Inhalte, die vom Browser dargestellt werden, sind im `<body>`-Bereich angegeben. Im Code-Beispiel 4 ist das eine Überschrift und ein Absatz und wird, wie in Abbildung .12 dargestellt, im Browser angezeigt.

Das ist eine Grosse Ueberschrift

Das ist ein Absatz

Abbildung .12.: HTML Grundgerüst

.1.1. Attribute

HTML-Tags können durch Attribute erweitert werden. Die Attribute werden innerhalb der spitzen Klammern angegeben, siehe Listing 5. Besonders wichtig sind globale Attribute, die für alle HTML-Elemente verwendet werden können. Mithilfe des globalen Attributs `class` können mehrere HTML-Tags in einer Kategorie beziehungsweise Klasse zusammengefasst werden. Die Eigenschaften dieser Klasse können im Anschluss in der CSS-Datei beeinflusst werden, siehe das folgende Kapitel.

```
1 <h1 class="Ueberschrift">
2     Das ist eine Grosse Ueberschrift
3 </h1>
```

Listing 5: HTML Attribute

.2. CSS

CSS werden verwendet, um

- dem HTML-Dokument einen ansprechenden Stil zuzuweisen,
- das Layout des HTML-Dokumentes zu definieren und
- das Layout so zu gestalten, dass sich der Inhalt automatisch an die Bildschirmgröße anpasst.

Generell gilt, dass mit HTML die Inhalte definiert werden und mit CSS das Aussehen. Die Syntax, um CSS-Eigenschaften für HTML-Elemente zu definieren, ist wie folgt aufgebaut.

```
1 selektor {
2     Eigenschaften : Wert;
3 }
```

Listing 6: Die generelle Syntax für CSS-Eigenschaften

Selektoren können HTML-Elemente wie `<h1>` und `<p>`, oder Klassen und IDs, wie bereits im Kapitel .1 angesprochen, sein. Wird eine Klasse als Selektor verwendet, muss ein Punkt vor den Namen geschrieben werden, bei einer ID eine Raute [W3S21a].

Um die Überschrift aus Listing 5 grün zu färben, würde die CSS-Syntax wie in Listing 7 dargestellt, aussehen. Das Ergebnis ist in Abbildung .13 dargestellt.

```
1  <!-- eine Klasse als Selektor -->
2  .Ueberschrift {
3      color : green;
4  }
5
6  <!-- ein HTML-Element als Selektor-->
7  h1 {
8      color : green;
9  }
```

Listing 7: Grüne Überschrift CSS

Das ist eine Grosse Ueberschrift

Das ist ein Absatz

Abbildung .13.: Grüne Überschrift

.2.1. CSS in die HTML-Datei einbinden

Wie im Kapitel .1 bereits erwähnt, ist es möglich, eine CSS-Datei im HTML-Head zu verlinken, siehe Listing 8. Darin wird die Datei `style.css` eingebunden.

```
1 <head>
2     <link rel="stylesheet" href="style.css">
3 </head>
4 <body>
5     <h1 class="Ueberschrift">
6         Das ist eine Grosse Ueberschrift
7     </h1>
8 </body>
```

Listing 8: CSS-Datei in HTML verlinken

Das Attribut `href` gibt den Pfad der verlinkten Ressource an. Die Beziehung des verknüpften Dokuments zum aktuellen Dokument wird mit dem Attribut `rel` angegeben [Mc21]. CSS Deklarationen können auch direkt in das HTML Dokument geschrieben werden, indem man den HTML-Tag `<style>` verwendet, Listing 9.

```
1 <head>
2   <style>
3     .Ueberschrift {
4       color : green;
5     }
6   </style>
7 </head>
8 <body>
9   <h1 class="Ueberschrift">
10    Das ist eine Grosse Ueberschrift
11  </h1>
12 </body>
```

Listing 9: CSS-Datei in HTML einbinden

3. JavaScript

JavaScript ist eine Programmiersprache, mit der man komplexe Funktionen für Webseiten realisieren kann. JavaScript findet immer dann Anwendung, wenn eine Internetseite mehr als nur statische Informationen darstellt [Mc21].

Mithilfe von JavaScript können:

- Informationen in Variablen gespeichert,
- Operationen auf Webinhalte, wie zum Beispiel Texte ausgeführt und
- auf Ereignisse reagiert werden, wie zum Beispiel auf einen Maus-Klick.

3.1. JavaScript in die HTML-Datei einbinden

JavaScript kann, genauso wie CSS, entweder unter Verwendung des HTML-Tags `<script>` direkt in das HTML-Dokument geschrieben werden, oder der Code wird als externe Datei eingebunden. Dafür wird auch der HTML-Tag `script` verwendet, mit dem Attribut `src1`, und dem Pfad der JavaScript Datei [W3S21c], siehe Listing 10.

¹ source (Quelle)

```
1  <head>
2      <!--direkte Einbindung -->
3      <script>
4          // JavaScript Code
5      </script>
6
7      <!-- als externe Datei Einbinden -->
8      <script src="dateiname.js"></script>
9  </head>
```

Listing 10: JavaScript in HTML einbinden