universität wien

MASTERARBEIT / MASTER'S THESIS

Titel der Masterarbeit / Title of the Master's Thesis

„Analyse und Evaluierung von Lösungsansätzen zur Offlinenutzung von Web Mapping Applikationen“

verfasst von / submitted by

Giacomo Klopfer

angestrebter akademischer Grad / in partial fulfilment of the requirements for the degree of

Master of Science (MSc)

Wien, 2022 / Vienna 2022

Studienkennzahl lt. Studienblatt / degree programme code as it appears on the student record sheet:	A 066 856
Studienrichtung lt. Studienblatt / degree programme as it appears on the student record sheet:	Kartographie und Geoinformation
Betreut von / Supervisor:	Univ.-Prof. i.R. Dipl.-Ing. Dr. Wolfgang Kainz

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

ANF	Anforderung
API	Application Programming Interface
CSS	Cascading Style Sheets
GIS	Geoinformationssystem
HTML	Hypertext Markup Language
HTTPS	HyperText Transfer Protocol Secure
JS	JavaScript
ÖK	Österreichische Karte
OL	Openlayers
PWA	Progressive Web App
SDK	Software Development Kit
SQL	Structured Query Language
SW	Service Worker
URL	Uniform Resource Locator
WMA	Web Mapping Applikation

Danksagung

Die vorliegende Arbeit entstand im Rahmen des Masterstudiums Kartographie und Geoinformation an der Universität Wien.

Ich möchte mich bei Herrn Univ.-Prof. i.R. Dipl.-Ing. Dr. Wolfgang Kainz für die Betreuung der Masterarbeit und bei Herrn Christian Mayer und Herrn Jakob Miksch der Firma Meggsimum für die Zusammenarbeit bedanken.

Erklärung

Hiermit versichere ich,

- dass ich die vorliegende Masterarbeit selbstständig verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt und mich auch sonst keiner unerlaubter Hilfe bedient habe,
- dass ich dieses Masterarbeitsthema bisher weder im In- noch im Ausland in irgendeiner Form als Prüfungsarbeit vorgelegt habe
- und dass diese Arbeit mit der vom Begutachter beurteilten Arbeit vollständig übereinstimmt.

Wien, am_____ _____

Unterschrift des/der Studierenden

Kurzfassung

Diese Masterarbeit beschäftigt sich mit der Offlinefunktionalität von Web Mapping Applikationen. In Zusammenarbeit mit dem Büro für Geoinformatik Meggsimum soll ermittelt werden, welche Technologien eine Web App offlinefähig machen und ob diese für Web Mapping Applikation geeignet sind. Der Fokus liegt dabei auf der Untersuchung, ob Erfassungstätigkeiten wie Kartierungen und Bestandsaufnahmen trotz einer mangelnden Internetverbindung weiterhin funktionieren. Zuerst werden drei Lösungsansätze aufgezeigt, die eine Offlinenutzung von Web Applikationen möglich machen. Beim ersten Ansatz handelt es sich um Progressive Web Apps, eine neue Art von Web Apps, die durch ein bestimmtes Tool eine Offlinefähigkeit herstellen können. Der zweite Ansatz ist die Verwendung von Apache Cordova, ein Native Wrapper, der einer Web App die Funktionen einer Native App bereitstellt. Zuletzt wird noch die Möglichkeit der Einrichtung eines lokalen Webservers auf einem mobilen Endgerät näher untersucht. Die drei Lösungen werden anhand einer eingehenden Literaturrecherche miteinander verglichen. Nach der Analyse wird anhand von Anforderungen, die im Rahmen eines Workshops in Zusammenarbeit mit der Firma Meggsimum erarbeitet wurden, evaluiert, inwiefern diese Technologien für eine Web Mapping Applikation passend sind und welche der Technologien am besten für eine prototypische Umsetzung einer Offline Web Mapping Applikation geeignet ist. Auf Grundlage dieser Untersuchungen wird deutlich, dass Progressive Web Apps aufgrund ihrer Aktualität, ihrer Nutzerfreundlichkeit und Plattformunabhängigkeit am besten für den praktischen Teil dieser Arbeit verwendbar sind. Für die Umsetzung wurde eine Web Mapping Applikation erstellt und zu einer Progressive Web App migriert. Es stellt sich heraus, dass die Offlinefunktionalität auch für Web Mapping Applikationen gut funktioniert und Progressive Web Apps in naher Zukunft Native Apps ersetzen können.

Abstract

This master thesis investigates the offline functionality of web mapping apps. In collaboration with the Büro für Geoinformatik Meggsimum it is determined, which tools can make web apps offline capable and if these tools are suitable for web mapping apps. The focus lies on the investigation, whether data collection like mapping and surveying is possible despite of an insufficient internet connection. In the process three solutions, which can make web apps offline usable, are compared. The first approach is a Progressive Web App, which is a new kind of web app and has a tool implemented that enables the offline functionality. The second approach is the use of Apache Cordova, a native wrapper, that can give web apps the functions of a native app. Finally, the possibility of an installation of a local webserver on a mobile device will be examined. The comparison of the three possible solutions is based on literature research. Based on requirements, that are worked out in a workshop with the firm Meggsimum, will be evaluated, if the technologies are suited for web mapping applications and which of them will be used for a prototypical implementation of an offline web mapping application. Based on these studies it is evident, that because of their timeliness, user-friendliness and their platform independency, Progressive Web Apps were the most suitable for the practical part of this master thesis. For the implementation a, for this thesis prebuild, Web Mapping Application will be migrated to a Progressive Web App. It turns out that the offline functionality also works fine for Web Mapping Applications and that Progressive Web Apps soon could replace native Apps on the market.

1 Einleitung

1.1 Problemstellung und Relevanz des Themas

Der moderne Mensch ist daran gewöhnt, ständig - sowohl in der Arbeitswelt als auch im Privaten - mit einem mobilen Endgerät (Smartphone, Tablet) mit dem Internet verbunden zu sein. Nach einer Studie [1] verfügen 89% Prozent der Deutschen über ein Smartphone, rund 94% dieser Smartphones sind täglich im Einsatz. Mobile Endgerätbenutzer*innen können immer mehr Dinge von unterwegs erledigen und sind nicht mehr auf die Internetverbindung zu Hause oder im Büro angewiesen. Durch die gesteigerte Mobilität und die mobilen Nutzungsmöglichkeiten wurde beinahe schon jede*r einmal mit dem Problem konfrontiert, dass im öffentlichen Personennahverkehr (ÖPNV) oder an abgelegenen Orten, im Einzugsgebiet von schlecht ausgebauten Netzen, die Verbindung zum Internet abbricht. Bei Web Mapping Applikationen[1] ist es wichtig, dass es beispielsweise an abgelegenen Orten mit keiner oder schlechter Internetverbindung zu keinem Verbindungsverlust kommt, damit die Kernfunktion dieser Anwendung weiterhin verfügbar ist. Die uneingeschränkte Nutzung auf einem mobilen Endgerät kann dann erst wieder fortgesetzt werden, wenn die Internetverbindung wiederhergestellt ist. Um eine Nutzung von Web Mapping Applikationen auch im Offlinezustand zu ermöglichen, stehen bestimmte Technologien zur Verfügung.

Im Rahmen dieser Masterarbeit wird ausgearbeitet, welche Lösungsansätze aktuell vorhanden sind und welche davon für Web Mapping Apps am besten geeignet sind. Dies soll anhand von erarbeiteten Kriterien analysiert und bestimmt werden. Anhand des Ergebnisses soll anschließend eine Beispielapp zu einer offlinefähigen Web Mapping Applikation migriert werden. Die Umprogrammierung der Applikation wird entsprechend ausführlich dokumentiert.

Diese wissenschaftliche Arbeit soll die mögliche Offlinefunktionalität von Web Mapping Applikationen analysieren, um einen Beitrag zur weiteren Forschung in diesem Themengebiet zu leisten. Das Hauptaugenmerk dieser Arbeit, die in Zusammenarbeit mit dem Lehrstuhl für Kartographie und Geoinformation der Uni Wien verfasst wird, liegt dabei auf dem geographischen Aspekt und soll sich vor allem auf die Funktionalität der GIS-Tools im Offlinemodus konzentrieren. Die Masterarbeit ist dahingehend relevant, dass nach aktuellem Forschungsstand die Offlinefunktionalität bei Web Mapping Applikationen im Bereich der Vermessung noch nicht eingehend untersucht wurde. In Zusammenarbeit mit dem Büro für Geoinformatik Meggsimum aus Mutterstadt, Deutschland, wurden drei Lösungsansätze erarbeitet, die untersucht werden sollen. Die Hauptkriterien sind, dass diese Open-Source[2] sind, offline funktionieren und Erfassungstätigkeiten ermöglichen. Anhand dieser Masterarbeit können sich Interessenten einen Überblick machen, um einen passenden Ansatz für ihre Applikation zu finden.

[1] Eine Applikation, die es dem User ermöglicht interaktiv mit einer Karte zu arbeiten
[2] Für jeden frei zugänglich

1.2 Zielsetzung

Ziel der vorliegenden Abschlussarbeit ist eine mögliche praktische Umsetzung einer Web Mapping Applikation, die offline nutzbar ist und eine Datenerfassung ermöglicht. Dazu zählt ein dokumentierter Workflow, der Interessenten eine vereinfachte Migration einer Web Mapping App in eine Offlineapp ermöglichen soll. Zur Bestimmung geeigneter Lösungsansätze sollen zuerst einige technische Möglichkeiten, die es aktuell auf dem Markt gibt, nach ihrer Funktionsweise untersucht werden. Anhand der gesammelten Daten werden die Technologien miteinander verglichen. Zur Bestimmung der am besten geeigneten Technologie, die im Praxisteil der Masterarbeit an der Beispielapplikation getestet werden soll, werden die Technologien anhand von Kriterien, die eine Web Mapping Applikation fordern soll, mit Hilfe einer Nutzwertanalyse evaluiert. Dadurch kann die Eignung dieser Technologien für Web Mapping Applikationen bestimmt werden.

Aus der Zielsetzung ergeben sich dann folgende Forschungsfragen:

„Welche Lösungsansätze für die Offlinenutzung von Web Applikationen gibt es und inwiefern sind diese für das Web Mapping geeignet?“

Dazu sollen noch folgende Arbeitsfragen beantwortet werden:

- *Welche technischen Möglichkeiten gibt es, um eine Applikation offline nutzbar zu machen?*
- *Inwiefern unterscheiden sich diese hinsichtlich ihrer Funktionsweise?*
- *Nach welchen Kriterien lassen sich diese vergleichen und evaluieren?*
- *Welcher Lösungsansatz ist für eine Web Mapping Applikation am besten geeignet?*

1.3 Methodisches Vorgehen

Im ersten Teil der Arbeit werden zuerst theoretische Grundlagen zum Themengebiet erläutert und es wird der aktuelle Stand der Forschung beschrieben. Unter Zuhilfenahme von Literatur soll erarbeitet werden, wie die zu verwendenden Technologien systematisch funktionieren und wie sich Unterschiede diesbezüglich erkennen lassen. Nach der Untersuchung werden die Ergebnisse bezüglich der Eignung für Web Mapping Applikationen mit Hilfe einer Nutzwertanalyse evaluiert. Damit soll schließlich bestimmt werden, welche dieser technischen Möglichkeit praktisch angewendet werden soll.

Im praktischen Teil der Arbeit wird die evaluierte Technologie an einer Web Mapping Applikation, die vorher kurz vorgestellt werden soll, veranschaulicht. Der praktische Vorgang wird im Verlauf der Arbeit ausgearbeitet. Zum Schluss wird das Ergebnis untersucht und bewertet.

1.4 Stand der Forschung

Nach einer eingehenden Literaturrecherche konnte festgestellt werden, dass das Thema der Offlinefunktionalität bei Web Applikationen seit der Einführung von Progressive Web Apps (PWA) sehr aktuell ist, aber im Forschungsbereich Web Mapping wissenschaftlich noch nicht eingehend erforscht wurde. Cross-Plattform Applikationen und Hybride Applikationen, die auch eine Offlinefunktionalität bieten, wurden bereits in mehreren wissenschaftlichen Arbeiten im Detail untersucht [2] [3] [4] [5]. Sie werden bezüglich ihrer Technologien und Frameworks eingehend verglichen und evaluiert. Auch eine allgemeine Gegenüberstellung unterschiedlicher Cross-Plattform Apps wurde durch einige Autoren schon durchgeführt [6] [7] . Mit dem Erscheinen der PWAs im Jahr 2015 wurden in den folgenden Jahren einige Papers verfasst, die PWAs näher untersuchen und bezüglich ihrer Leistung im Vergleich zu Cross-Plattform Applikationen und Native Apps diskutieren [8]. Hierbei geht es auch um die Performance, den Energieverbrauch und die Sicherheit von PWAs [9] [10] [11]. Weiter werden PWAs auch in Hinsicht auf die Eignung für bestimmte Unternehmen untersucht. Neben der wissenschaftlichen Literatur gibt es aufgrund der Aktualität des Themas viele Internetquellen [12] [13] und Artikel [14] [15] [16]. Auch einige Bücher, die dabei helfen eine PWA zu entwickeln oder eine App zu einer PWA zu migrieren, besprechen dieses Thema [17] [18] [19]. Die Benutzung eines lokalen Webserver auf einem mobilen Gerät wurde in diesem Zusammenhang wissenschaftlich noch nicht untersucht.

Die Literaturrecherche ergab, dass es viele Interessierte gibt, die sich mit dem Thema beschäftigen und schon zahlreiche Studien ausgeführt wurden. Dennoch steht bei keiner dieser Papers das Thema Offlinefunktionalität im Zusammenhang mit Web Mapping wirklich im Mittelpunkt. Dies soll in dieser Abschlussarbeit geschehen. Dabei werden nicht nur PWAs näher betrachtet, sondern auch zwei weitere mögliche Technologien, die eine Offlinefunktionalität erreichen. Neben der Analyse der Technologien wird vor allem das Augenmerk auf den Prozess der Umsetzung zu einer offlinefähigen Web Mapping Applikationen gelegt. Dabei soll untersucht werden, inwiefern sich dieses Vorgehen vom generellen Ablauf unterscheidet und ob es vor allem für Web Mapping Applikation geeignet ist.

1.5 Aufbau der Arbeit

Im **Kapitel 1** wird durch einleitende Worte beschrieben, was die Problemstellung der Arbeit ist, welches Ziel der Autor mit dieser Abschlussarbeit erreichen will, was der aktuelle Stand der Forschung ist und nach welcher Methode vorgegangen wird, um die Forschungsfrage zu beantworten.

Im **Kapitel 2** sollen technologische Grundlagen dem Leser helfen, das Thema der Arbeit besser zu verstehen. Hier werden die Begriffe Web Mapping und Web GIS näher erläutert und verschiedene Arten von Applikation dargestellt.

Das **Kapitel 3** beschreibt die Methodik der Masterarbeit. Dabei wird auf den durchgeführten Workshop, die Bewertung der Kriterien sowie die Umsetzung der Web Mapping Applikation eingegangen.

Das **Kapitel 4** befasst sich anschließend mit den drei zu untersuchenden Lösungsansätzen. Hierbei handelt es sich um die Progressive Web Apps, den Native Wrapper sowie den lokalen Webserver. Hier werden diese im Detail untersucht und der grundlegende Vorgang der Entwicklung wird erklärt.

Im **Kapitel 5** werden diese Ansätze anhand von erarbeiteten Kriterien verglichen und mit Hilfe einer Nutzwertanalyse beurteilt und anschließend bezüglich der Eignung für Web Mapping Applikation evaluiert.

Im **Kapitel 6** findet die Umsetzung einer Progressive Web App statt. Hierbei wird eine gewöhnliche Web Mapping App verwendet, die extra für diese Abschlussarbeit entwickelt wurde. Anhand dieser App wird die Umsetzung der Offlinefunktionalität einer Web Mapping App veranschaulicht.

Im **Kapitel 7** wird die Masterarbeit mit einem Fazit zur Ausarbeitung der Problemstellung und einem Ausblick zusammengefasst.

2 Theoretische Grundlagen

In diesem Kapitel werden die für diese Masterarbeit relevanten theoretischen Grundlagen aufgezeigt. Hierbei werden zuerst der Begriff Web Mapping und Web GIS aufgezeigt, sowie Beispielapplikationen in Österreich. Im zweiten Teil des Kapitels werden die verschiedenen Arten von Applikation vorgestellt.

2.1 Web Mapping und Web GIS

2.1.1 Begriffserklärung

Es gibt verschiedene Definitionen für den Begriff Web Mapping:

Neumann beschreibt Web Mapping als "the process of designing, implementing, generating and delivering maps on the World Wide Web." ([20], S.567)

Dorman behauptet: "With the advent of web mapping, geographical information can be shared, visualized,and edited in the browser." ([21], S.12)

Web Mapping ist also eine Interaktion mit einer Karte im Web.

In Verbindung zum Web Mapping wird auch oft der Begriff Web GIS erwähnt. Dieser wird dem Web Mapping fälschlicherweise oft gleichgestellt. Der Unterschied ist, dass dem Nutzer bei einem Web GIS mehr Interaktions- und Handlungsmöglichkeiten zur Verfügung stehen. Die Grenze zwischen Web Mapping und Web GIS ist undeutlich und Web GIS konzentriert sich mehr auf die Analyse sowie auf die Aufbereitung von Geodaten. [20]

Die Beispielapplikation in dieser Arbeit lässt sich als eine Web Mapping App beschreiben. Neben der Veranschaulichung der Offlinefunktionalität bei einer Web Mapping App soll die Applikation dazu dienen, eine interaktive Karte zu präsentieren, die zusätzlich zu der Standard Zoom-Funktion noch das Hinzufügen von einigen Attributen erlaubt, nämlich Punkten, Linien und Polygonen. Nach den beschriebenen Definitionen entspricht sie also einer Web Mapping App, da sie nicht zu sehr auf die Verarbeitung von Geodaten konzentriert ist. Nichtsdestotrotz ist eine Erweiterung zu einem Web GIS für weitere Erfassungstätigkeiten möglich.

2.1.2 Beispielapplikationen in Österreich

In folgendem Kapitel sollen beispielhaft einige Web Mapping Applikationen vorgestellt werden, die aktuell auf dem Markt zu finden sind.

(1) Austrian Map online (www.austrianmap.at)

Austrian Map online wird vom Bundesamt für Eich- und Vermessungswesen (BEV) gepflegt, das für die Kartographie, Geoinformatik, Eich- und Messwesen zuständig ist. Das Portal ermöglicht den Zugriff auf die ÖK 500, ÖK 250, ÖK 50 und besitzt neben der

standardmäßigen Zoomfunktion auch die Möglichkeit der Distanzmessung und der Koordinatenbestimmung.

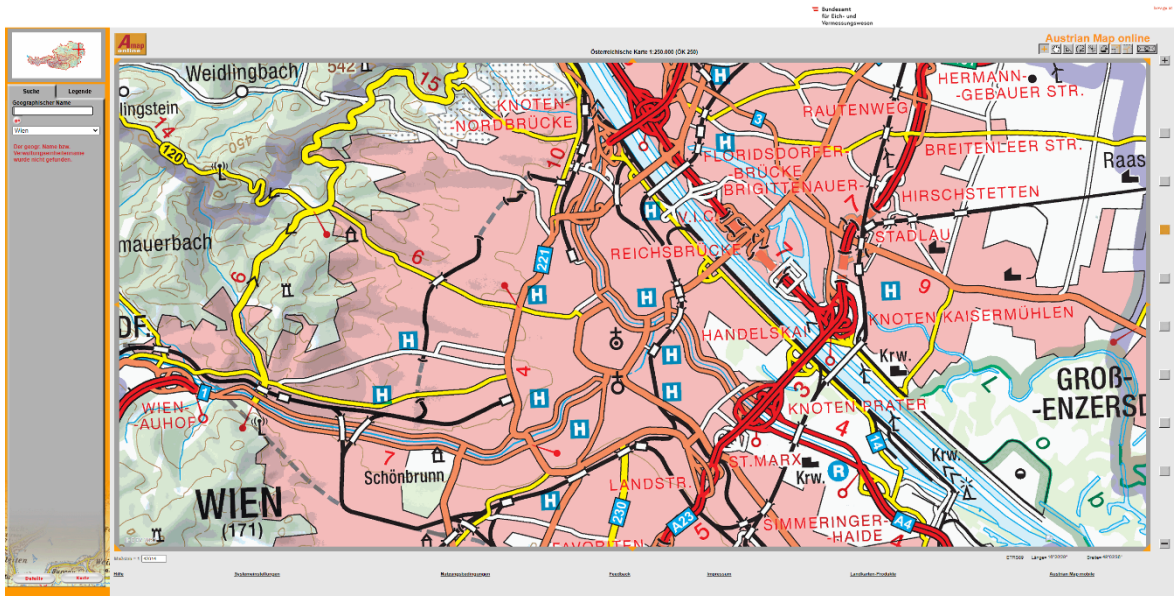


Abbildung 1: Austrian Map Online

(2) Geoportale der Bundesländer

Jedes Bundesland in Österreich verfügt über ein Geoportal, das eine statische oder interaktive Karte online anbietet. Die Web Applikation von Salzburg[3] kann eher als Web GIS angesehen werden, da sie zusätzlich zu den standardmäßigen Funktionen Werkzeuge für die Datenverarbeitung bereitstellt (vgl. Abbildung 2).

[3] https://www.salzburg.gv.at/sagismobile/sagisonline/map/Basiskarten/Alle%20Themen

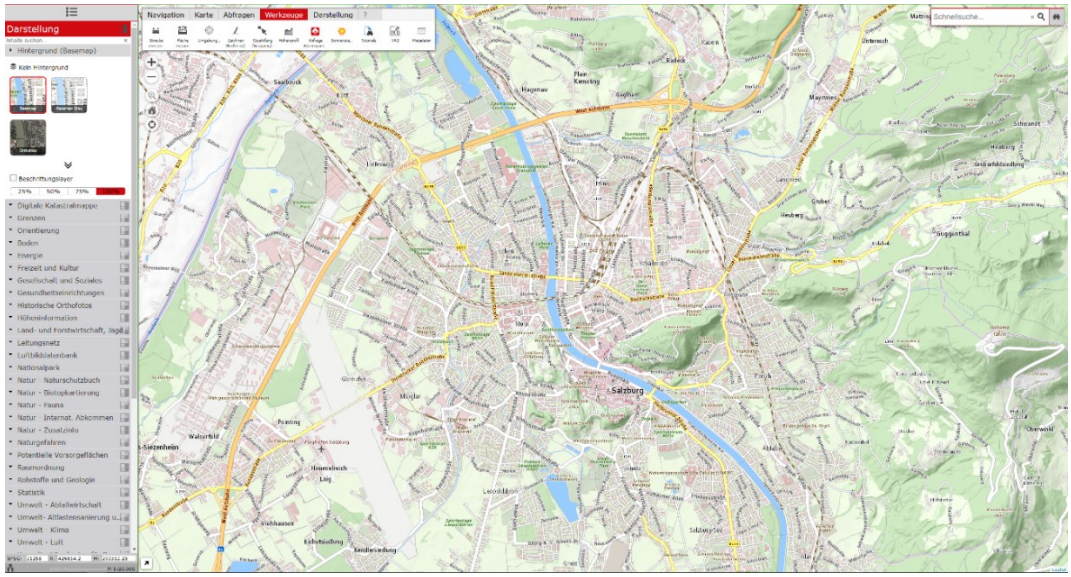


Abbildung 2: Web GIS von Salzburg

(3) Geodatenviewer der Stadtvermessung Wien (Vienna GIS)[4]

Der Geodatenviewer ist im Gegensatz zu den anderen vorgestellten Web Apps eine Anwendung zu rein visuellen Zwecken. Sie veranschaulicht verschiedene Karten, Luftbilder, Geländedarstellungen, Oberflächenmodelle und Geodaten. Außerdem ermöglicht sie dem Nutzer Daten kostenfrei von der Plattform herunterzuladen.

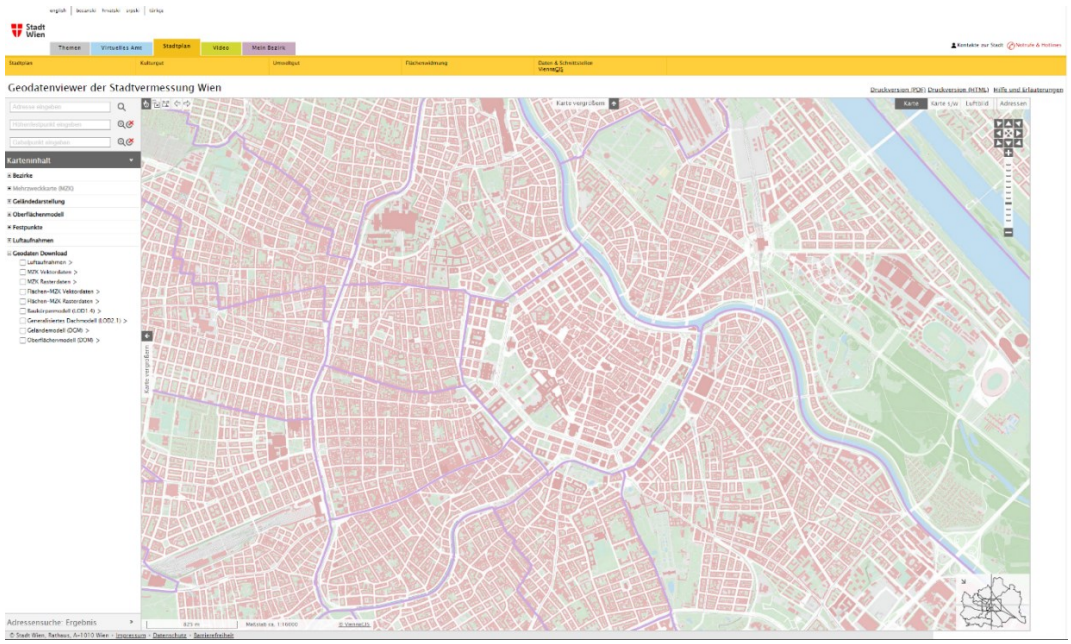


Abbildung 3: Vienna GIS

[4] https://www.wien.gv.at/ma41datenviewer/public/ (Datenquelle: Stadt Wien – data.wien.gv.at)

2.2 Apps

2.2.1 Native Apps

Native Apps sind Applikationen, die aus einem App Store kostenlos oder kostenpflichtig heruntergeladen und auf dem entsprechenden mobilen Gerät installiert werden können. Die bekanntesten führenden Betriebssysteme sind Android, iOS und Windows [22]. Native Apps sind dabei „stets für eine bestimmte Plattform entwickelt und entsprechen üblicherweise auch den Konventionen der Benutzeroberfläche dieser Plattform“ ([23], S.58) Bei Android ist es der Google Play Store, bei Apple der Apple Store und bei Windows der Microsoft Store. Eine Installation von Applikationen ohne Download aus einem offiziellen App Store ist auch möglich, jedoch stellt dieses Verfahren ein Sicherheitsrisiko dar, da sie von Drittanbietern kommen. Der Installationsvorgang erfolgt hierbei direkt über die Installationsdatei, die durch einen Downloadvorgang erworben wird [24]. Updates der Native Apps werden ebenfalls wieder aus dem jeweiligen App Store bezogen. Abhängig vom Betriebssystem werden Native Apps in verschiedenen Programmiersprachen geschrieben. Java wird für Android benötigt, Objective-C für iOS und das .NET Framework für Windows Phone [25]. Damit eine Native App für mehrere Betriebssysteme kompatibel ist, muss sie an das neue Betriebssystem angepasst werden und es muss ein neuer Programmcode programmiert werden. Dies ergibt sich als ein Nachteil im Vergleich zu anderen Apps in Hinsicht auf den benötigten Zeitaufwand sowie die erhöhten Produktionskosten. Ein Vorteil ist die Möglichkeit des ungehinderten Zugriffs auf die Hardware sowie die Sensoren des mobilen Gerätes. [25]

In Abbildung 4 werden die Betriebssysteme bezüglich ihres technologischen Hintergrunds kurz vorgestellt:

	Apple iOS	Android	Blackberry OS	Windows Phone
Languages	Objective-C, C, C++	Java (some C, C++)	Java	C#, VB.NET and more
Tools	Xcode	Android SDK	BB Java Eclipse Plug-in	Visual Studio, Windows Phone development tools
Packaging format	.app	.apk	.cod	.xap
App stores	Apple App Store	Google Play	Blackberry App World	Windows Phone Marketplace

Abbildung 4: Übersicht der meistbenutzten mobilen Betriebssysteme [24]

2.2.2 Web Apps

Web Apps sind im Gegensatz zu Native Apps nicht auf dem Gerät installiert. Sie sind wie eine Webseite über eine URL abrufbar und basieren auf den Webtechnologien HTML, CSS und JavaScript. Das Icon, das auf dem Gerät erscheint, entspricht einem Lesezeichen zur entsprechenden Web App. Wie der Begriff schon sagt, funktioniert eine herkömmliche Web App nur über eine Internetverbindung. Außerdem ist es nur beschränkt möglich, auf User-Interface-Elemente wie Hardware- und Softwarekomponenten zuzugreifen. Sie können von jedem mobilen Endgerät aus plattformunabhängig über den größten Teil der Browser

aufgerufen werden. Dies hat den Vorteil, dass die Apps von verschiedenen Betriebssystemen ohne Einschränkung verwendet werden können. Im Vergleich zu den Nativen Apps verringern sich dadurch unter anderem der Aufwand und die Kosten bei der Entwicklung der Applikation. Auch ein Vertrieb über einen App-Store entfällt bei dieser Art von Applikation. [26]

2.2.3 Hybrid Apps

Hybride Apps sind Apps, die sowohl eine Web- als auch eine Native Applikation darstellen. Über ein Framework kann sich eine Web App wie eine Native App verhalten und somit Zugriff auf Hardware- und Softwarekomponenten des Geräts erhalten. Dies geschieht in der Form, dass ein sogenannter Wrapper die Web App umhüllt und sie als eine Native Applikation darstellt. Hybride Apps werden somit auch über den App-Store bezogen und über diesen aktualisiert [4]. Durch das Framework wird es der Web App ermöglicht auch offline zu funktionieren [27]. Dies funktioniert, da es der Applikation möglich ist, auf dem Gerät Daten abzuspeichern. Bekannte Frameworks sind zum Beispiel Ionic, Capacitor oder Apache Cordova, welche noch im Detail im Verlauf dieser Arbeit untersucht werden (vgl. Kapitel 4.2.1).

2.2.4 Cross-Plattform-Apps

Cross-Plattform-Apps werden einmal programmiert und werden anschließend für mehrere Betriebssysteme optimiert. Der Entwicklungsaufwand ist damit erheblich geringer als der einer Nativen App und spart dem Entwickler Zeit und Kosten. Die Applikation besitzt auch die Möglichkeit, auf Hardware- und Softwarekomponenten zuzugreifen. Bekannte Frameworks sind React Native, Titanium Mobile und Flutter SDK. [28]

	Native Apps	**Web Apps**	**Hybrid Apps**	**Cross-Plattform-Apps**
Bezug	App Store	Webbrowser	App Store	App Store
Internetverbindung	Nicht notwendig	Notwendig	Nicht notwendig	Nicht notwendig
Hardwarezugriff	Vollständig	Nein	Teilweise	Teilweise
Softwarezugriff	Vollständig	Nein	Teilweise	Teilweise
Datenspeicherung	Lokal	Cache	Lokal	Lokal
Programmiersprachen	Java	HTML, CSS, JavaScript	HTML, CSS, JavaScript	HTML, CSS, JavaScript

Tabelle 1: Eigenschaften der verschiedenen Apps (Eigene Darstellung)

3 Methodik

Das Hauptziel dieser Masterthesis ist die Untersuchung, ob Technologien, die einer Web App eine Offlinefunktionalität verleihen können, auch für eine Web Mapping Applikation geeignet sind. Im folgenden Kapitel wird im Detail das methodische Vorgehen für eine Entwicklung einer offlinefähigen Applikation erläutert. Nach der Beschreibung des Workshops wird auf die durchgeführten Interviews und den Vergleichsvorgang der Lösungsansätze eingegangen. Die Evaluation der Technologien wird anhand einer Nutzwertanalyse durchgeführt. Aus dem Ergebnis wird abgeleitet, dass die Entwicklung einer PWA ein geeigneter Lösungsansatz ist. Dieser wird zur Vollendung der Masterarbeit anhand einer Beispielapp umgesetzt.

Abbildung 5: Methodisches Vorgehen

3.1 Workshop

In Mutterstadt wurde bei Meggsimum mit Herrn Christian Mayer und Herrn Jakob Miksch ein eintägiger Workshop abgehalten mit dem Ziel Offlineapps zu untersuchen und Anforderungen zu erarbeiten, die eine Offline Web Mapping App erfüllen soll.

3.1.1 Interview und Gruppendiskussion

Um einige Anhaltspunkte zum Thema Offlinefunktionalität in Bezug auf Web Mapping zu bekommen, wurde zur weiteren Untersuchung des Themengebiets ein problemzentriertes Interview durchgeführt. Hierbei handelt es sich um ein leitfadengestütztes Interview, das jedoch frei gestaltet werden kann. Das Gespräch wird in verschiedene Themenblöcke aufgeteilt, zu denen jeweils eine offene Frage sowie bei Bedarf spezifizierende Fragen gestellt werden. Das problemzentrierte Interview wird durchgeführt, wenn der Forscher sein Vorwissen, das er sich mit Hilfe einer Literaturrecherche angeeignet hat, um eine persönliche Meinung erweitern möchte. [29]

Beim Interview wurden folgende Fragen gestellt:

- Wie funktioniert die Offlinefähigkeit bei Web Applikationen?
- Welche offlinefähigen Applikationen kennen Sie bereits und wie funktionieren diese?
- Welchen Nutzen sollen die Offlineapplikationen haben?
- Welche sind die Zielgruppen?
- Für welche Endgeräte sollten Offlineapplikationen entwickelt werden?
- Welche Funktionen sollten Offlineapplikationen anbieten?
- Wie erfolgt das Datenmanagement?

Im Anschluss an das Interview wurden im Rahmen einer Gruppendiskussion die Ergebnisse näher betrachtet und kategorisiert. Eine Gruppendiskussion ist, ebenso wie das Interview, eine qualitative Erhebungsmethode, bei der mehrere Teilnehmer über ein Thema diskutieren. [30] Im Rahmen dieser Masterarbeit dient sie dazu die Erkenntnisse aus den Interviews kritisch zu betrachten.

Aus der Gruppendiskussion haben sich folgende Fragestellungen ergeben (vgl. Tabelle 2):

	Fragestellungen
Usecase	Wie lassen sich die Anforderungen an die Offlineapplikation technisch umsetzen?
	Welche Open-Source Software kommt für eine Implementierung in Frage?
	In welcher Netzwerkumgebung ist die Offlinefunktionalität möglich?
Datenmanagement	Wie erfolgt die Datenspeicherung innerhalb der Applikation?
	Lassen sich die Offlinedaten in der Applikation bei Wiederverbindung zum Internet synchronisieren?
	Welche Datenformate werden von der Applikation unterstützt?
	Wie können aufgenommene Daten verwaltet werden?

Funktionalität	Welche Funktionen unterstützt die Applikation im Offlinezustand?
	Mit welchen Endgeräten sind Offlineapplikationen kompatibel?
	Welche Betriebssysteme werden unterstützt?
	Wie ist der Performanceunterschied im Vergleich zum Onlinezustand?
Setup	Kann die Applikation offline aktualisiert werden?
	Welcher Aufwand ist für die Einrichtung einer Offlineapplikation notwendig?

Tabelle 2: Fragestellungen im problemzentrierten Interview (Eigene Darstellung)

3.1.2 Ergebnisse

Bei der Gruppendiskussion wurden nach der Erarbeitung der Fragestellungen vier Themengebiete besprochen: die Anwendung der Applikation, das Datenmanagement, die Funktionalität und das Setup. Bei der Anwendung der Applikation standen die Zielgruppe und der Nutzen sowie die Bereitstellung der App im Vordergrund. Bei der Datenspeicherung ging es um die Art und Größe des möglichen Speichers und die Datenformate. Im Bereich der Funktionalität wurden mögliche Funktionen erarbeitet. Zuletzt wurden die Aktualisierungsmöglichkeiten sowie der allgemeine Aufwand zur Einrichtung einer Offline Web Mapping Applikation besprochen.

Anhand der erarbeiteten Fragestellungen konnten die Anforderungen (vgl. Abbildung 6) definiert werden, welche der Analyse und Evaluation von Lösungsansätzen in Bezug auf die Forschungsfragen dienen. Die Kriterien wurden anhand der vier Kategorien Usecase, Datenmanagement, Funktionalität und Setup sortiert. Im nächsten Schritt wurden Lösungsansätze für eine Offlinefunktionalität untersucht, die aktuell auf dem Markt existieren und noch regelmäßig durch den Hersteller aktualisiert werden. Eine fundierte Marktrecherche hat ergeben, dass die Technologien PWA, Native Wrapper und lokaler Webserver am besten für eine detailliertere Analyse in Frage kommen. Mögliche Umsetzungen einer PWA wurden anhand einer eingehenden Literaturrecherche ausgearbeitet. Die drei Technologien geben einen Überblick über den aktuellen Standard bezüglich der Offlinefunktionalität.

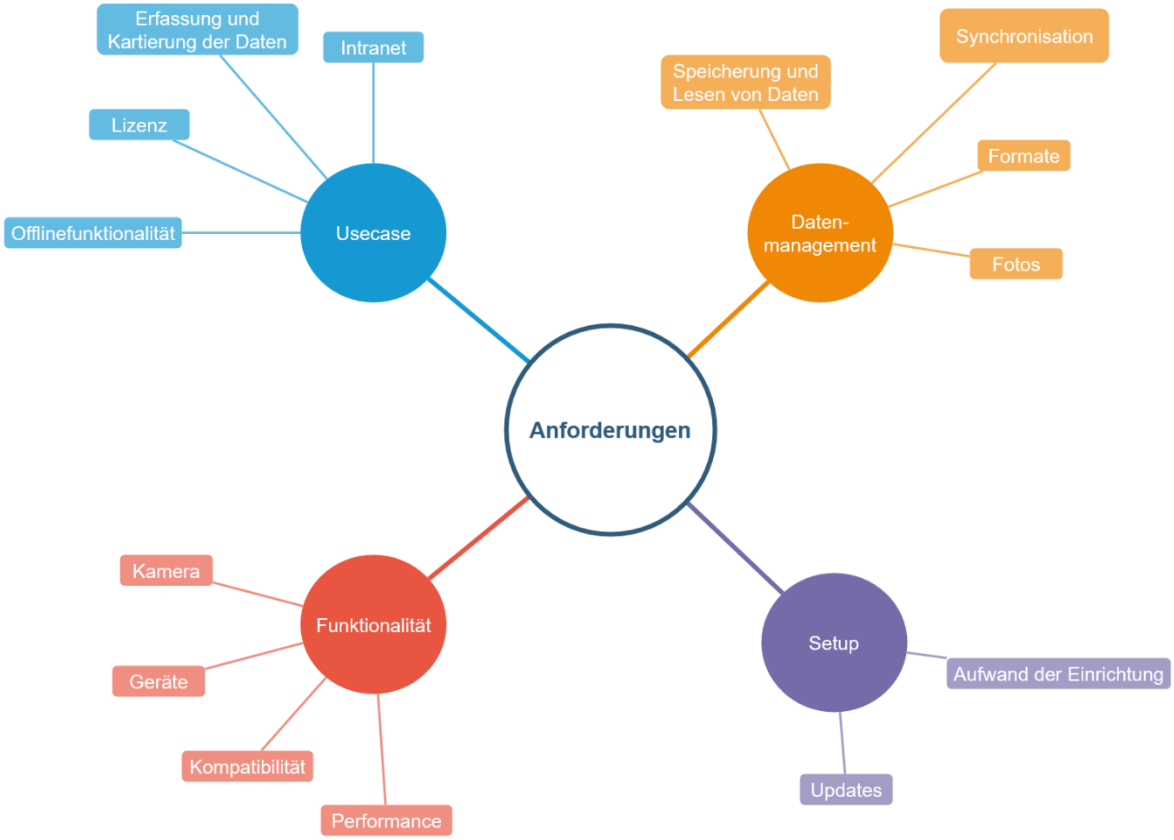


Abbildung 6: Übersicht der Eigenschaften einer Offline Web Mapping Applikation (Eigene Darstellung)

3.1.3 Anforderungen

Die Anforderungen, die im Workshop erarbeitet wurden, werden in folgendem Kapitel nach Rupp [31] mit Anforderungsschablonen dokumentiert.

In Abbildung 7 und Abbildung 8 sind zwei Schablonen aufgeführt, die den Aufbau zeigen. Die Schablonen dienen der übersichtlichen Darstellung der Bedeutung der zu untersuchenden Anforderungen. In Kapitel 5 wird nach dem Vergleich der Technologien untersucht, inwiefern die Bedingungen erfüllt wurden. Die Begriffe zwischen den spitzen Klammern werden je nach Anwendungsfall angepasst. Die Schlagwörter "muss", "sollte" und "wird" werden in Tabelle 2 definiert. Abbildung 7 unterscheidet sich in der Hinsicht, dass der Schablone aus Abbildung 8 noch eine Bedingung vorangestellt ist.

Begriffe	Bedeutung
muss	Die Anforderung ist verpflichtend und muss vorhanden sein
sollte	Die Anforderung ist wünschenswert, muss aber nicht vorhanden sein
wird	Die Anforderung ist verpflichtend für die Zukunft

Tabelle 3: Definition der Schlüsselwörter (vgl. [31], S.168)

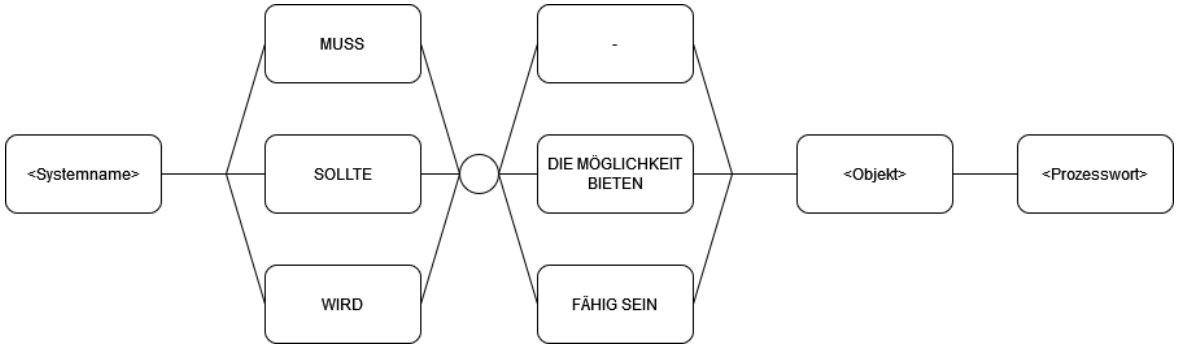


Abbildung 7: Anforderungsschablone ohne Bedingung (vgl. [31], S.220)

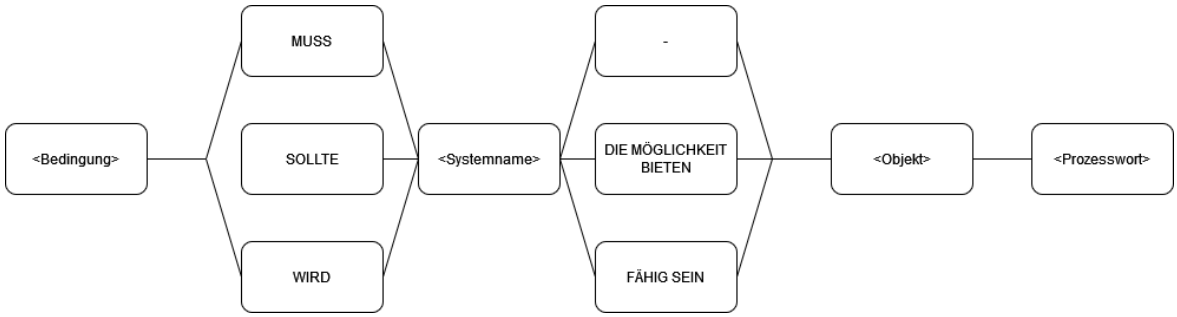


Abbildung 8: Anforderungsschablone mit Bedingung (vgl. [31], S.224)

Basierend auf den Schablonen werden die Anforderungen im Folgenden ausformuliert:

3.1.3.1 Usecase

Erfassungstätigkeiten:	Sobald der User offline ist, muss ihm die WMA die Möglichkeit bieten, Daten zu erfassen und zu kartieren.
Open-Source Lizenz:	Die WMA muss eine Open-Source-Lizenz haben.
Offlinefunktionalität:	Die WMA muss offline funktionieren.
Funktionalität im Intranet:	Sobald der User offline ist, sollte die WMA auch im Intranet funktionieren.

3.1.3.2 Datenmanagement

Speichern und Lesen von Daten: Sobald der User offline ist, muss die WMA fähig sein, Daten zu speichern und zu lesen.

Synchronisation: Nach dem Offlinezustand, sollte die WMA fähig sein, die neu hinzugefügten Daten zu synchronisieren.

Unterstützte Formate: Die WMA muss mehrere Datenformate lesen und schreiben können.

Speichern und Verlinken von Fotos: Sobald der User offline ist, sollte die WMA die Möglichkeit bieten Fotos zu speichern und fähig sein, diese zu verlinken.

3.1.3.3 Setup

Update: Die WMA sollte dem User die Möglichkeit bieten, selbstständig Updates durchzuführen.

Aufwand: Die Einrichtung der WMA sollte benutzerfreundlich sein.

3.1.3.4 Funktionalität

Performance: Sobald der User offline ist, sollte die WMA mit einer guten Performance funktionieren.

Kompatibilität: Die WMA sollte mit mehreren Betriebssystemen kompatibel sein.

Geräte: Die WMA muss fähig sein, auf mobilen Plattformen zu funktionieren.

Kamera: Sobald der User offline ist, sollte die WMA dem User die Möglichkeit bieten, Fotos zu machen.

3.2 Vergleich und qualitative Bewertung der Anforderungen

Die Anforderungen werden im nächsten Schritt miteinander verglichen und anschließend anhand einer Prioritätenanalyse und einer Nutzwertanalyse qualitativ bewertet und evaluiert. Der Ansatz, der am Ende die beste Bewertung erhält, wird im praktischen Teil dieser Arbeit umgesetzt.

3.2.1 Nutzwertanalyse

"Ziel der Nutzwertanalyse ist es, verschiedene komplexe Lösungsalternativen in Abhängigkeit zu den Präferenzen des Entscheidungsträgers und dem daraus abgeleiteten systematischen Zielsystem in eine Rangfolge zu bringen." [32]

Sie ist eine qualitative Bewertungsmethode, die Verwendung findet, wenn nicht-monetäre Aspekte in die Bewertung einbezogen werden sollen und wenn eine rein quantitative Bewertung nicht sinnvoll ist oder zu keinem Ergebnis führt. Der Gesamtnutzwert, der am Ende der Analyse gebildet wird, entscheidet über die Qualität der Technologien. Die Erfüllung der Anforderungen hängt damit von der Größe des Gesamtnutzwertes ab. [32]

Für eine erfolgreiche Nutzwertanalyse muss folgende Vorgehensweise systematisch eingehalten werden [32]:

Vorgehensweise:

1) Festlegung und Gewichtung der Bewertungskriterien anhand einer Prioritätenanalyse
2) Beurteilung der Lösungsansätze
3) Berechnung der Nutzwerte
4) Evaluation

3.2.1.1 Festlegung und Gewichtung der Bewertungskriterien anhand einer Prioritätenanalyse

Für eine Bewertung von Lösungsansätzen sind Kriterien notwendig, nach denen diese verglichen werden können. Für die Bewertung werden die Anforderungen verwendet, die im Workshop erarbeitet wurden. Aufgrund der Oberflächlichkeit der festgelegten Kriterien werden diese nochmals unterteilt. Damit werden zuerst die Unterkriterien miteinander verglichen. Anschließend findet mit den Ergebnissen die gesamte Bewertung statt. Für ein besseres Verständnis der Anforderungen werden diesen Fragestellungen zugeteilt, nach denen die Beurteilung durchgeführt wird.

	Bewertungskriterien	Fragestellungen
ANF 1	**Lizenz**	Läuft die Applikation über eine Opensource Lizenz?
ANF 2	**Offlinefunktionalität**	Wie ist die Offlinefunktionalität der Applikation zu beurteilen?
ANF 3	**Speichern und Lesen von Daten**	
	Speichern von Daten	Wie groß ist der Datenspeicher?
	Lesen von Daten	Wie ist der Prozess beim Lesen der Daten zu beurteilen?
ANF 4	**Erfassungstätigkeiten**	
	Datenerfassung	Wie ist die Erfassung von Daten über die Applikation zu beurteilen?
	Kartierung	Wie ist die Kartierung von Daten über die Applikation zu beurteilen?
ANF5	**Kompatibilität**	Mit wie vielen Betriebssystemen ist die App kompatibel?
ANF6	**Geräte**	Von wie vielen Geräten wird die App unterstützt?
ANF7	**Performance**	
	Ladezeit	Wie lang lädt die Applikation im Offlinezustand?
	Energieverbrauch	Wie hoch ist der Energieverbrauch?
ANF8	**Aufwand**	
	Dokumentation	Wie ausführlich ist die Einrichtung und Konvertierung dokumentiert?
	Software	Ist besondere Software notwendig?
	Hardware	Ist eine leistungsstarke Hardware notwendig?
	Arbeitszeit	Mit welcher Arbeitszeit ist bei der Einrichtung zu rechnen?
	Vorkenntnisse	Inwieweit muss der Programmierer in die Materie eingearbeitet sein?
ANF 9	**Unterstützte Formate**	Wie viele Datenformate werden akzeptiert?
ANF10	**Update**	Wie benutzerfreundlich ist das Updaten der Applikation?
ANF 11	**Fotos**	
	Speicherung	Ist das Speichern von Fotos möglich? Wie viele Fotos können gespeichert werden?
	Verlinkung	Ist die Verlinkung der Fotos mit den Messungen in der Karte möglich?
ANF12	**Kamera**	Wie ist die Einrichtung einer Kamera zu beurteilen?
ANF 13	**Synchronisation**	Funktioniert der Austausch zwischen dem Gerät und dem Intranet?
ANF 14	**Intranet**	Wie gut funktioniert die Offlinefunktionalität im Intranet?

Tabelle 4: Bewertungskriterien für die Nutzwertanalyse (Eigene Darstellung)

Die ermittelten Kriterien sind unterschiedlich in ihrer Wichtigkeit und müssen deswegen für die Bewertung gewichtet werden. Die Gewichtung wird mit Hilfe der Prioritätenanalyse durchgeführt. Anhand dieser Analyse wird eine Rangfolge mehrerer Kriterien durch Gewichten geformt. In einer Präferenzmatrix (vgl. Abbildung 10) werden alle Anforderungen jeweils miteinander verglichen und bewertet. Die Bewertung findet anhand der Punktvergabe in der zweiten Spalte der Tabelle in Abbildung 9 statt, um den Gesamtwert "0" zu vermeiden. Die Noten stehen dabei für 1 = weniger wichtig, 2 = gleich wichtig und 3 = wichtiger. [32]

	Punktvergabe für die Kriterien A : B		
	Vergabe von 2 Punkten je Paar (Standard)	Vergabe von 4 Punkte je Paar (bspw. um zu vermeiden, dass Gf=0, wenn ein Kriterium immer nachrangig ist)	Vergabe von 4 Punkten je Paar (bspw. um größere Differenzierungen zu erreichen)
A ist deutlich wichtiger als B			4 : 0
A ist wichtiger als B	2 : 0	3 : 1	3 : 1
A und B sind gleich wichtig	1 : 1	2 : 2	2 : 2
A ist weniger wichtig als B	0 : 2	1 : 3	1 : 3
A ist deutlich weniger wichtig als B			0 : 4

Abbildung 9: Punktvergabe bei einer Präferenzmatrix (Quelle: [32])

Aus der Summe der Einzelbewertungen kann die Wichtigkeit der einzelnen Anforderungen in Prozent berechnet werden. Diese ergibt sich aus der Division der Summe der Einzelbewertungen durch die Gesamtsumme aller Bewertungen (vgl. Abbildung 10). Die Gewichtung wird nach der Beurteilung der Kriterien angewendet. [32]

Kriterien	A	B	C	D	Summe je Kriterium	Gewichtungs-faktor Gf (%)
A		2	2	2	6	50,0
B	0		2	2	4	33,3
C	0	0		1	1	8,3
D	0	0	1		1	8,3
SUMME					12	100

Abbildung 10: Beispiel einer Präferenzmatrix (Quelle: [32])

3.2.1.2 Beurteilung der Lösungsansätze und Berechnung der Nutzwerte

Bei diesem Schritt wird in der Nutzwertanalyse beurteilt, inwieweit die Lösungsansätze die gestellten Anforderungen erfüllen. Dafür wird als Bewertungsmaßstab eine Skala von 1 (sehr niedrig) bis 5 (sehr hoch) verwendet (vgl. Tabelle 5). Die Beurteilung der Lösungsansätze basiert auf der zur Verfügung stehenden Literatur und auf dem subjektiven Empfinden.

Zielerfüllungfaktor	**1**	**2**	**3**	**4**	**5**
Bewertung	sehr niedrig	niedrig	mittel	hoch	sehr hoch

Tabelle 5: Bewertungsskala (Eigene Darstellung)

3.2.1.3 *Evaluation*

Im letzten Schritt der Nutzwertanalyse wird das Ergebnis evaluiert. Dabei werden die Notengebung, die unterschiedlichen Punktzahlen und die am besten bewertete Technologie kommentiert. Der Lösungsansatz mit dem höchsten Gesamtnutzwert wird anschließend für die Umsetzung einer Offline Web Mapping Applikation angewendet.

3.3 Umsetzung der Web Mapping App

Im praktischen Teil der Masterarbeit wird der Lösungsansatz umgesetzt, der am besten den erarbeiteten Anforderungen entspricht. Hierbei wird im Detail der Entwicklungsprozess erläutert, der von der Programmierung der Beispielapp über die Konvertierung zu einer Offlineapp bis hin zum Starten der Applikation führt. Im Anschluss wird der praktische Teil evaluiert und mögliche Schwierigkeiten und Herausforderungen werden aufgezeigt.

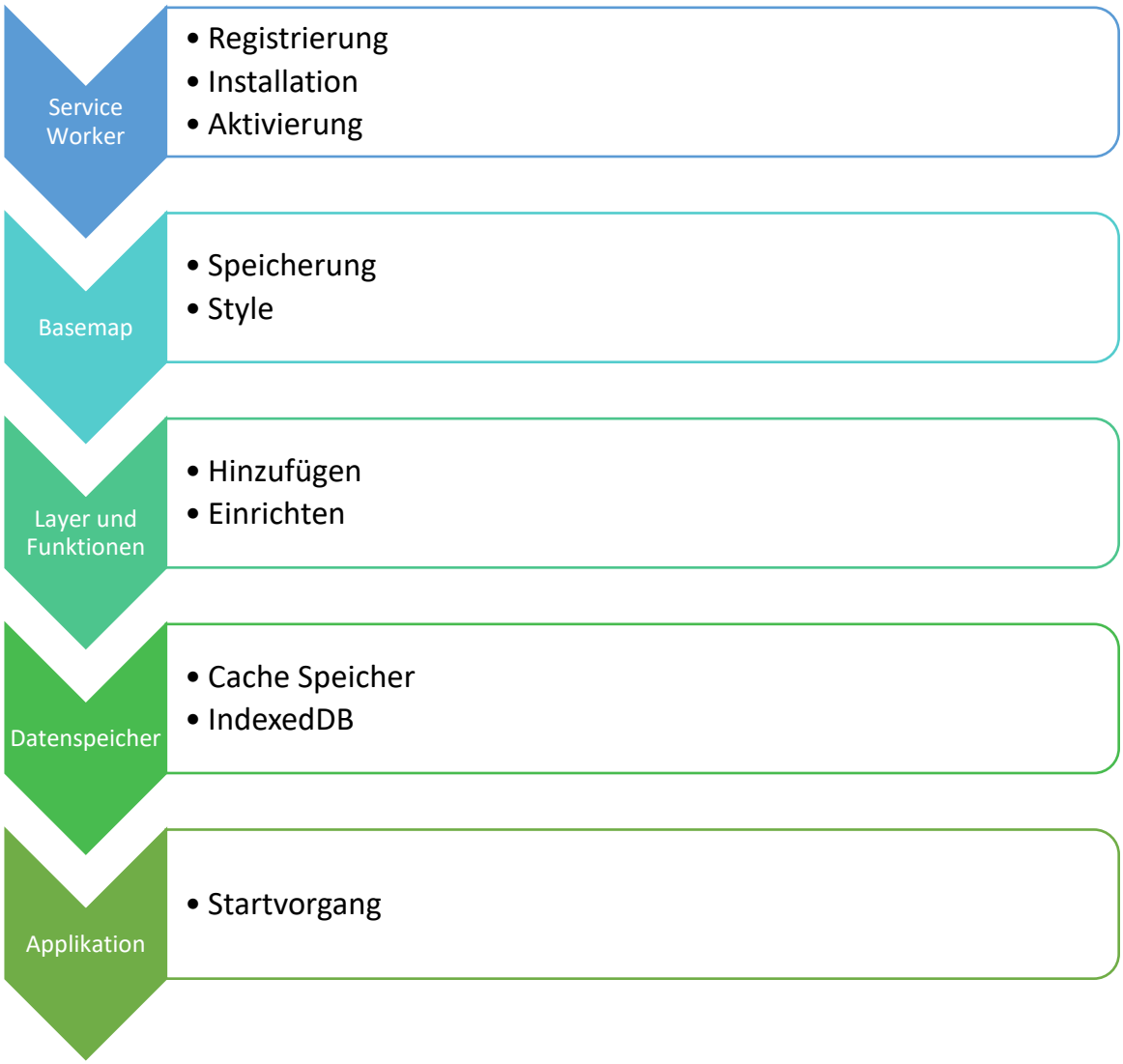


Abbildung 11: Prozess zur Umsetzung der Web Mapping App (Eigene Darstellung)

4 Lösungsansätze

In diesem Kapitel werden die verschiedenen Arten von Ansätzen erläutert, die eine Offlinefähigkeit von Web Mapping Apps erreichen. Nach einer allgemeinen Erläuterung soll mehr auf die Offlinefunktionalität eingegangen werden, welche in dieser Masterarbeit den Fokus erhält. Dem Leser soll aufgezeigt werden, wie die klassisch erstellten Applikationen zur Offlinefähigkeit kommen und welche Tools benötigt werden, um diese zu erreichen.

Diese Untersuchung basiert auf dem Offline-First Prinzip. Offline-First bedeutet, dass eine Applikation so programmiert wird, dass sie sowohl online als auch offline verwendet werden kann [33] [34]. Wenn eine Applikation offline ist, beschreibt das den Zustand, bei dem der Browser nicht mit dem Internet kommunizieren kann. Weitere Funktionen, die eine Netzverbindung benötigen, wie zum Beispiel Positionierungsdienste, können weiterhin funktionieren. Um eine weitere Verwendung der Applikation gewährleisten zu können, sind Datenbanken notwendig, auf denen Dateien gelagert werden müssen, um auf diese im Offline-Zustand wieder zugreifen zu können [33]. Die Technologien sowie die benötigten Datenbanken werden in den nächsten Unterkapiteln noch näher erläutert.

4.1 Progressive Web Apps

Als erste dieser Technologien werden die Progressive Web Apps vorgestellt. Auf eine allgemeine Beschreibung folgt eine detailliertere Betrachtung der Funktionsweise. Dazu zählen vor allem die Technik, die sich hinter einer PWA befindet und die Voraussetzungen für eine einwandfreie Funktionalität. Zuletzt werden einige aktuelle Beispielapps kurz aufgeführt.

4.1.1 Allgemeine Beschreibung

Der Begriff Progressive Web Apps wurde zum ersten Mal im Jahr 2015 von Alex Russel und Frances Berriman benutzt [35]. Die Technologie ist in Zusammenarbeit von Google, Apple, Microsoft, Mozilla und weiteren Technologiefirmen entstanden, wobei sie aktuell vor allem von Google weitergeführt wird [36]. Progressive Web Apps sind eine Kombination aus Native Apps und Web Apps (vgl. Kapitel 2.2).

„Progressive Web Apps (PWA) are built and enhanced with modern APIs to deliver enhanced capabilities, reliability, and installability while reaching anyone, anywhere, on any device with a single codebase.“ [13].

Durch die Eigenschaften Leistung, Zuverlässigkeit und Installierbarkeit ähneln sie plattformspezifischen Applikationen. Die Leistung beschreibt in diesem Zusammenhang die Fähigkeit der App, Dienste anzubieten, die bisher nur bei plattformspezifischen Applikationen zu sehen waren. Zuverlässigkeit steht für eine einwandfreie User Experience, die durch eine uneingeschränkte Nutzung der Applikation auch bei Netzwerkstörungen erreicht wird. Die dritte Eigenschaft, Installierbarkeit, zeigt die Möglichkeit der Installation der Applikation auf dem Gerät auf. [13]

Progressive Web Apps lassen sich durch folgende Eigenschaften noch konkreter beschreiben [37]:

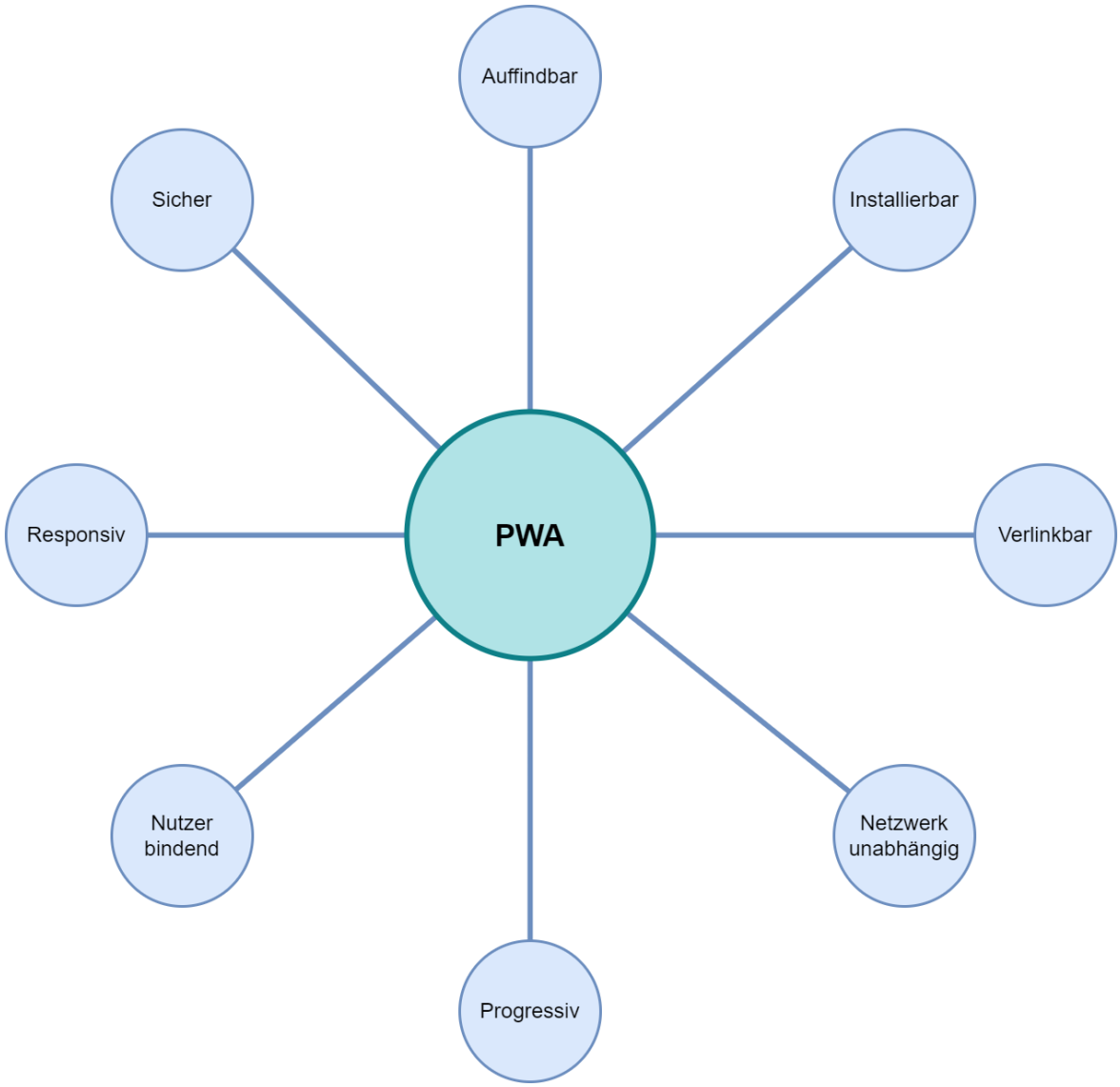


Abbildung 12: Eigenschaften von Progressive Web Apps (Eigene Darstellung)

Auffindbar –Sie können über eine URL oder eine Suchmaschine gefunden werden. Durch das Web App Manifest wird die PWA zu einer Applikation, die nach Bedarf dann auch über einen App Store downloadbar wird.

Installierbar – Sie sind als Applikation auf dem Home Screen des Endgerätes installierbar. Wie bei einer Native App wird hier ein Icon auf dem Bildschirm erstellt wodurch der Nutzer beim nächsten Zugriff direkt auf die PWA zugreifen kann.

Verlinkbar – Da sie über eine URL aufgerufen werden, können diese mit anderen Personen geteilt werden, ohne dass diese die App installieren müssen.

Netzwerk unabhängig – Durch den Service Worker ist eine Progressive Web App nicht an eine Internetverbindung gebunden und funktioniert damit auch offline. Durch die Speicherung von bereits genutzten Daten in einem Cache ist im Offlinezustand ein einfacher Zugriff auf diese möglich.

Progressiv – Diese Eigenschaft bedeutet, dass eine PWA sich den aktuellen Gegebenheiten anpassen kann. Das heißt, wenn einige Funktionen der PWA mit dem verwendeten Browser nicht kompatibel sind, führt das nicht automatisch zum Beenden der App, sondern es funktionieren dann nur noch die passenden Features.

Nutzer bindend – Durch die Push Notifications kann der Nutzer auch nach Beenden der Applikation noch über Updates oder Benachrichtigungen in Kenntnis gesetzt werden.

Responsiv – Die Apps können unabhängig vom Format auf allen Endgeräten sowie auf mobilen Endgeräten wie Smartphones oder Tablets und auf dem Desktop verwendet werden.

Sicher – Eine PWA lässt sich nur über HTTPS öffnen und ist damit sicher vor Cyberangriffen.

4.1.2 Service Worker

Der Service Worker ist der entscheidende Faktor, der einer PWA die Offlinefunktionalität ermöglicht. Es ist ein Tool, das als Proxy zwischen einer Applikation, die im Browser durchgeführt wird und dem Webserver fungiert. Ein Proxy ist in diesem Fall eine Art Vermittler im Netzwerk zwischen dem Client (Nutzer) und dem Webserver. Der Service Worker wird eigenständig durchgeführt und ist nur aktiv, wenn er benötigt wird. [38]

Bei einer Anfrage an den Webserver durch den Client im Offlinemodus wird diese vom Service Worker abgefangen und anschließend weiterverarbeitet. Dieser verarbeitet die Anfrage und ruft, wenn vorhanden, die angefragten Daten aus einem Speicher ab. Diese Speichermöglichkeiten werden gleichzeitig mit dem Service Worker installiert. Nach dem Laden der Daten aus dem Speicher schickt dieser eine Antwort an den Client, die aus den angefragten Daten besteht. Wenn sich die angefragten Dokumente nicht im Speicher befinden, wird dem Client im Falle einer Netzwerktrennung eine vorgefertigte HTML-Seite auf dem Gerät dargestellt. [38]

Der Service Worker, das Web App Manifest (vgl. Kapitel 4.1.3) und das HTTPS Protokoll verwandeln eine Web App zu einer Progressive Web App. Der Nutzer hat die Möglichkeit die Applikation weiterhin trotz Offlinezustand zu benutzen [39]. Um einen Zugriff auf die persönlichen Daten durch externe Skripte zu verhindern, ist ein Service Worker auf eine Domain limitiert und wird nur über HTTPS oder über einen Localhost gestartet [38]. Der *Scope* beschreibt eine Kombination aus der Webseite und dem Pfad, wodurch theoretisch auch mehrere voneinander getrennte Progressive Web Apps über denselben Ordner betrieben werden können. [38]

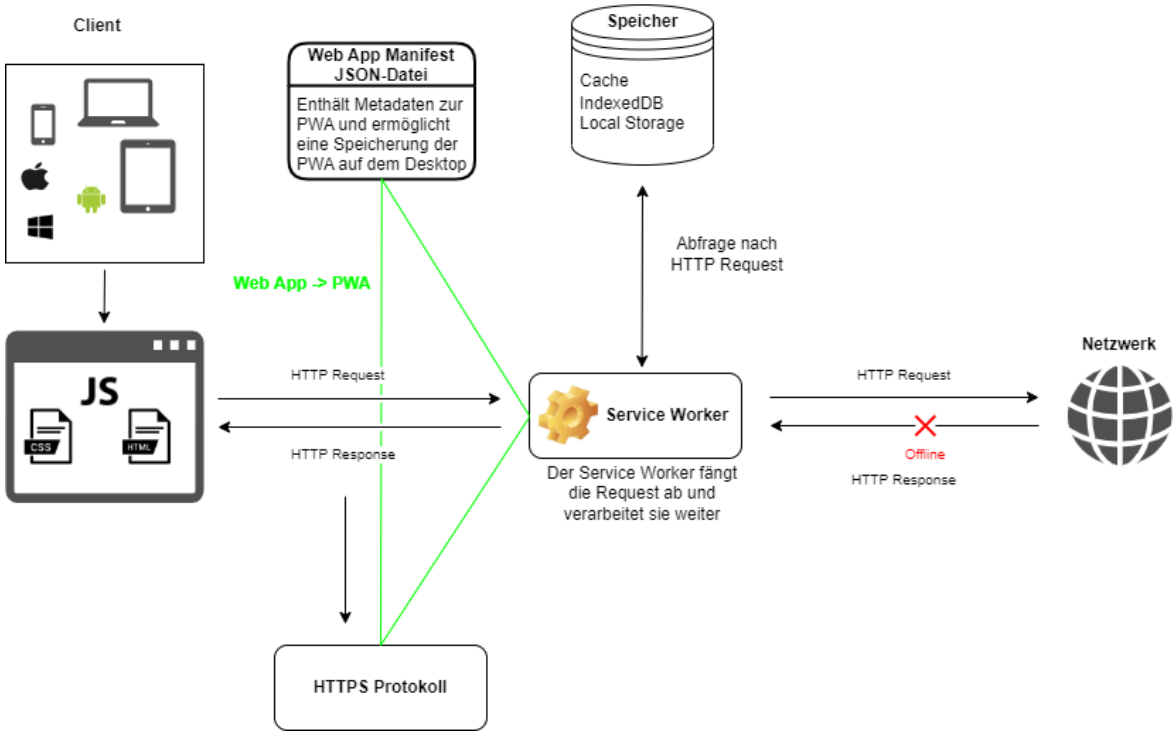


Abbildung 13: Funktionalität eines Service Worker (Eigene Darstellung)

Der Service Worker durchgeht einen „Lebenszyklus“, bevor er benutzt werden kann. Folgendes Diagramm soll den Prozess der erstmaligen Installation des Service Workers veranschaulichen (vgl. Abbildung 14).

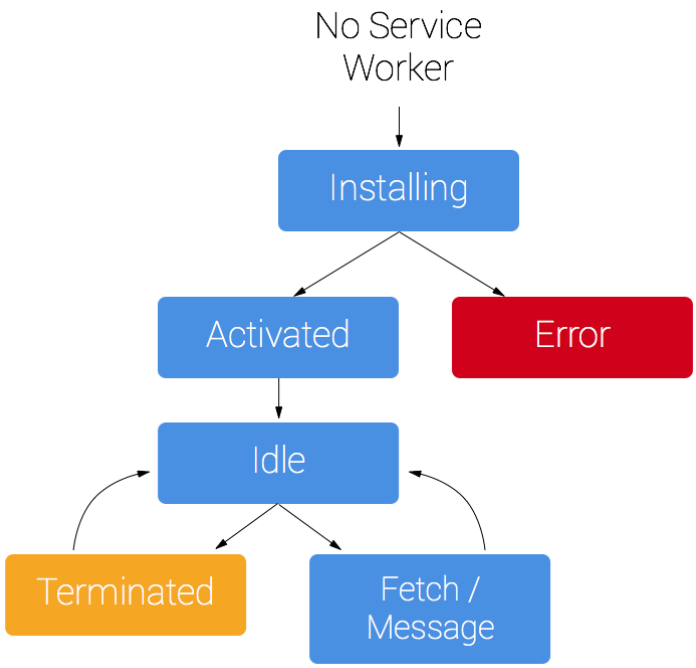


Abbildung 14: The Service Worker Lifecycle (Quelle: [19], S.134)

Damit bei einer gewöhnlichen Web App der Service Worker zum Einsatz kommen kann, wird dieser zuerst installiert. Nach der Installation muss er aktiviert werden. Dies geschieht durch ein einfaches Neuladen der Web App. Sollte die Aktivierung fehlschlagen, wird dem Nutzer eine Fehlermeldung erscheinen. Um das Problem zu beheben, kann entweder der Fehler durch Neuladen der App oder durch Verbesserung des Programmcodes gelöst werden. Einmal aktiviert, läuft der Service Worker im Hintergrund in einem separaten Thread ab ("Idle") und wird aktiv, wenn er benötigt wird. Dies geschieht, wenn der Nutzer eine Anfrage stellt und der Service Worker diese Anfrage abfängt ("Fetch"). Nach der Bearbeitung der Anfrage wird die Technologie wieder in den Ruhezustand versetzt oder kann beendet werden. [40]

Die Programmierung dieses Ablaufs wird im Detail im Kapitel 6 besprochen, wenn es um die Umsetzung einer PWA geht.

4.1.3 Web App Manifest

Das Web App Manifest beinhaltet in einem JSON-Textfile[5] die Metadaten[6] zu einer Web Applikation und bewirkt die Installation der Progressive Web App auf dem Gerät des Nutzers. Dadurch gelangt der Nutzer schneller zur Web App und erhält damit eine Native App User Experience. Im Web App Manifest sind unter anderem Name, Autor, Icon der App und eine Beschreibung vorhanden. [41]

[5] Die JavaScript Object Notation (JSON) ist ein Datenformat, das den Datenaustausch zwischen Anwendungen ermöglicht

[6] Metadaten enthalten Information über andere Daten, wie Dokumente oder Dateien

```
{
  "short_name": "PWMA",
  "name": "Progressive Web Mapping App",
  "icons": [
    {
      "src": "/img/logo.png",
      "type": "image/png",
      "sizes": "192x192"
    }
  ],
  "start_url": "/index.html?source=pwa",
  "background_color": "#3367D6",
  "display": "standalone",
  "scope": "/",
  "theme_color": "#3367D6",
  "shortcuts": [
    {
      "name": "Maps",
      "short_name": "Maps",
      "description": "Openlayersmap",
      "url": "/map?source=pwa",
      "icons": [{ "src": "/img/logo.png", "sizes": "192x192" }]
    }
  ],
  "description": "Openlayers_Background",
"screenshots": [
  {
    "src": "/img/logo-background.png",
    "type": "image/png",
    "sizes": "720x540"
  }
]
}
```

Abbildung 15: Codebeispiel des Web App Manifest aus der PWA

Mit "short_name" wird entschieden, welcher Name unter dem Icon stehen soll und "name" beschreibt die Bezeichnung der Applikation. Unter "icons" werden die Quelle des Icon Bildes, der Typ sowie die Größe eingetragen. Im nächsten Absatz werden die URL ("start_url") der Web Applikation sowie die Hintergrundfarbe ("background_color") festgelegt. Über "display" wird bestimmt, wie die Applikation dargestellt werden soll. In diesem Fall soll sie als eigenständige Applikation gezeigt werden. Unter "shortcuts" wird ein Link zu einer Seite eingefügt, die innerhalb der Web Applikation geöffnet werden soll. In diesem Fall handelt es sich um die Karte. Auch hier werden "name" und "short_name" sowie eine Beschreibung festgelegt. Anschließend werden noch die URL zur Karte sowie die Quelle für das Icon und dessen Größe genannt. Zuletzt folgt die Nennung des Hintergrundes der Applikation. Dieser muss ebenso mit Quelle, Dateityp und Größe beschrieben werden. [41]

Nach der Konfiguration der Web App Manifest Datei ist es dem Nutzer möglich, die PWA mit dem Aufpoppen eines Downloadsymbols im oberen rechten Quadranten des Browserfensters zu installieren (vgl. Abbildung 16). In Zukunft kann über das Icon, das auf dem Homescreen gespeichert wurde, schnell auf die Web Mapping Applikation zugegriffen werden. [41]

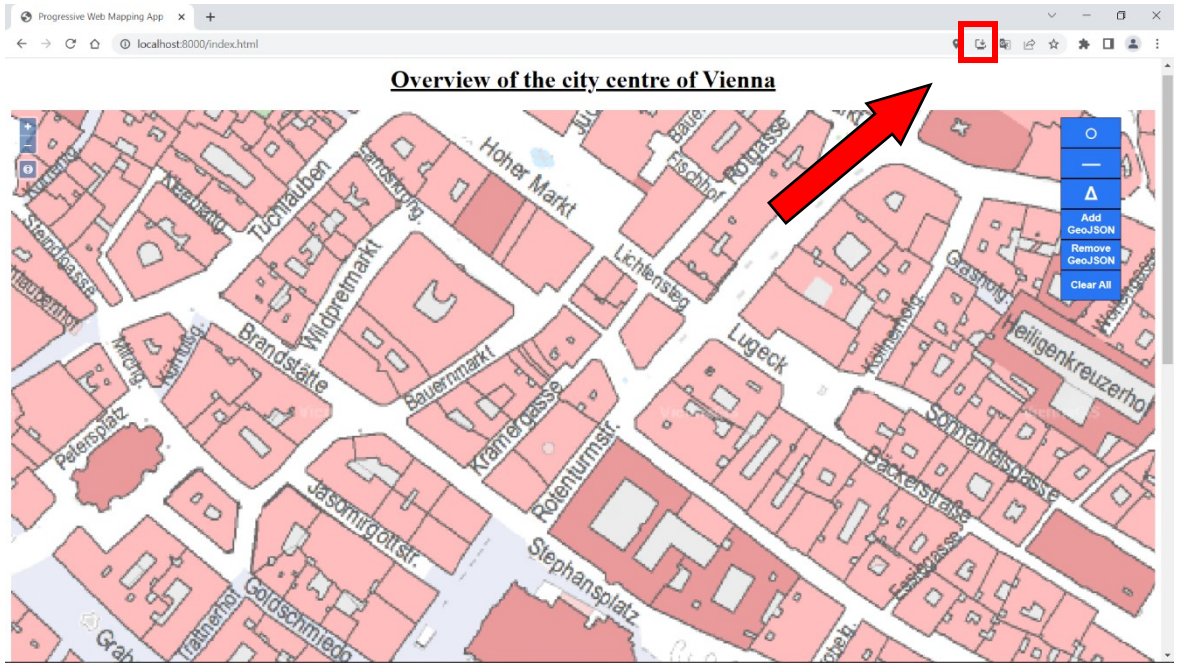


Abbildung 16: Symbol für den Download der PWA

4.1.4 Datenspeicherung

Zur Offlinenutzung müssen die Daten, die bei einer getrennten Verbindung zur Verfügung stehen sollen, in einer Datenbank gespeichert werden. Dafür gibt es verschiedene Speichermöglichkeiten.

- Der **Cache Storage** wird vor allem dazu verwendet, statische Ressourcen wie die JavaScript, CSS oder HTML-Dateien zu speichern, da nur eine sehr begrenzte Speicherkapazität zur Verfügung steht.[7]
- **WebSQL** ist eine SQL-Datenbank, die jedoch schon veraltet ist und nicht mehr aktualisiert wird.[8]
- **Web Storage** bietet mit **sessionStorage** und **localStorage** zwei Schlüssel-Wert-Speicher. SessionStorage stellt nur temporär für die Zeit des Seitenbesuchs einen Speicher bereit und localStorage stellt auch nach Schließen des Browsers einen durchgehenden Speicher bereit. Für Service Worker ist diese Speichermöglichkeit jedoch ungeeignet, da der Storage synchron arbeitet, was vom Service Worker nicht unterstützt wird [42].
- **IndexedDB** besitzt eine große Speicherkapazität und ist damit für große Mengen an Daten und Dateien gut geeignet. Sie ist eine JavaScript-basierte objekt-orientierte Datenbank. Sie arbeitet asynchron und ist somit auch für Service Worker geeignet [43].

Die relevanten Speichermöglichkeiten für die Umsetzung einer offlinefähigen Applikation sind das Cache Storage und IndexedDB. Diese werden im praktischen Teil der

[7] https://web.dev/learn/pwa/caching/
[8] https://www.w3.org/TR/webdatabase/

Abschlussarbeit im Detail besprochen. WebSQL und Web Storage werden nicht weiter untersucht.

4.1.5 PWA-Beispiele

Um zu verdeutlichen, wie aktuell das Thema der Progressive Web Apps ist und wie diese eingesetzt werden können, werden im Folgenden einige Beispiele von bekannten Firmen aus verschiedenen Branchen vorgestellt. Diese haben schon eine PWA eingeführt bzw. sind aktuell dabei eine zu entwickeln.

Twitter hat mit der PWA Twitter Lite eine Zunahme der Tweets um 75% erlangt. Web Push Nachrichten werden auch angezeigt, wenn die Applikation geschlossen ist.[9] Facebook befindet sich noch in der Entwicklung und ist noch nicht auf allen mobilen Endgeräten verfügbar[10.]

Die größte B2B-Handelsplattform[11] der Welt Alibaba.com hat ihre Native App zu einer Progressive Web App migriert. Durch die PWA erlangte das Unternehmen mehr mobile Nutzer sowie einen viermal höheren Zugriff auf die App über den Home Screen.[12]

Starbucks ermöglicht Kunden wie bei einer installierten App auch mit der PWA das Menü anzuschauen sowie Bestellungen aufzugeben. Durch die PWA wird dem Konsumenten ermöglicht auch in Gegenden mit einer schlechten Internetverbindung die App zu benutzen.[13]

Die Hotelsuche trivago erreichte durch die Migration ihrer Native App zu einer PWA einen Buchungsanstieg von 97%.[14]

[9] https://pwa.bar/twitter/
[10] https://pwa.bar/progressive-web-app-facebook/
[11] Business-to-Business (B2B) ist die Geschäftsbeziehung zwischen zwei oder mehreren Unternehmen
[12] https://web.dev/alibaba/
[13] https://pwa.bar/starbucks/
[14] https://pwa.bar/trivago/

4.2 Native Wrapper

Die zweite Technologie, die in dieser Arbeit untersucht werden soll, ist der Native Wrapper. Hierbei handelt es sich um ein Framework, das eine bestehende Web Applikation zu einer hybriden App macht. Wie aus dem Begriff "Native Wrapper" ableitbar, umhüllt (engl. "to wrap") das Framework eine Web App und verleiht ihr damit Funktionen einer Native App. Der genaue Prozess wird im Laufe des Kapitels näher erläutert. Zuerst wird eine Marktanalyse durchgeführt, um dem Leser aufzuzeigen, welche Frameworks im Moment auf dem Markt sind. Anschließend findet eine Erläuterung der Architektur und der Funktionsweise eines ausgesuchten Native Wrappers statt.

4.2.1 Marktanalyse

Für die folgende Marktanalyse wurden Kriterien bestimmt und für einen Vergleich von drei Frameworks benutzt. Einer Marktanalyse in diesem Fachgebiet entsprechend, sind die Hauptkriterien die Kosten einer Einrichtung, die Kompatibilität und die Nutzbarkeit. Folgende Tabelle (vgl. Tabelle 6) soll dem Leser einen Überblick über die drei Native Wrapper geben:

Kriterien / **Frameworks**	Apache Cordova	Capacitor	Electron
Geräte	Mobile, Desktop (veraltet)	Mobile, Desktop	Desktop
Betriebssysteme	Android, iOS, Windows (veraltet)	Android, iOS, Windows	Windows, macOS, Ubuntu
Kosten	Open Source	Open Source	Open Source
Architektur	Ausführung durch Einbettung des HTML codes in einen Web View (vgl. Abbildung 18)	Ausführung durch Einbettung des HTML codes in einen Web View (vgl. Abbildung 18)	Ausführung durch Einbettung von Chromium & Node.js
Datenspeicher	LocalStorage, IndexedDB, Plugins	Capacitor Storage API, SQLite	LocalStorage, IndexedDB
Update	Plugin für ein automatisches Update oder manuell	Manuell oder über die Installation einer App (Appflow)	Entwickler muss Chromium & Node.js selbstständig aktualisieren
Zugriff auf native Schnittstellen	Möglich	Möglich	Möglich

Support	Apache Software Foundation (ASF)	Ionic	GitHub

Tabelle 6: Vergleich der Frameworks Apache Cordova, Capacitor und Electron (Eigene Darstellung)

Die Frameworks sind alle über den Desktop ausführbar. Jedoch sind nur Apache Cordova und Capacitor mit mobilen Geräten kompatibel. Da es vor allem das Ziel ist, dass die Applikationen unterwegs offline funktionieren, sollten diese am besten auch mobil funktionieren. Abgesehen von Electron unterstützen die Frameworks alle die gleichen Betriebssysteme. Die drei Frameworks sind Open Source und damit kostenfrei zugänglich. Auch bei der Architektur lassen sich Gemeinsamkeiten zwischen Cordova und Capacitor erkennen. In beiden Frameworks wird eine HTML-Datei eingebettet, wodurch die Web App zu einer Hybrid App wird. Bei Electron wird die Applikation durch Einbettung von Chromium und Node.js ausgeführt. Chromium ist ein Open-Source-Browser, in dem die größten Teile des Quellcodes von Google Chrome enthalten sind[15]. Da es in dieser Arbeit um Web Mapping Apps geht, sollten diese einen dementsprechend großen Speicherplatz haben. Zum Beispiel können Raster- und Vektorkacheln einen große Speichermenge aufbrauchen. In diesem Fall ist zu empfehlen, die Daten lokal auf dem Gerät zu speichern mit der Voraussetzung, dass dieses genug Platz anbietet. Bezüglich des Updates lässt sich bei Apache Cordova ein Plugin installieren, das die Applikation automatisch aktualisiert. Eine Alternative dazu wäre die Aktualisierung manuell durchzuführen. Bei Capacitor kann für automatische Updates eine zusätzliche App installiert werden oder eigenständig durchgeführt werden. Electron ist weniger nutzerfreundlich, da das Update nur durch den Entwickler durchgeführt werden kann. In allen drei Fällen ermöglichen die Frameworks, dass eine Web Applikation Zugriff auf native Schnittstellen erhält. Dies macht sie zu einer Hybrid App. Wie in der Tabelle dargestellt, findet der Support bei Apache Cordova durch die Apache Software Foundation (ASF), bei Capacitor durch Ionic und bei Electron durch Github statt. [44] [45] [46]

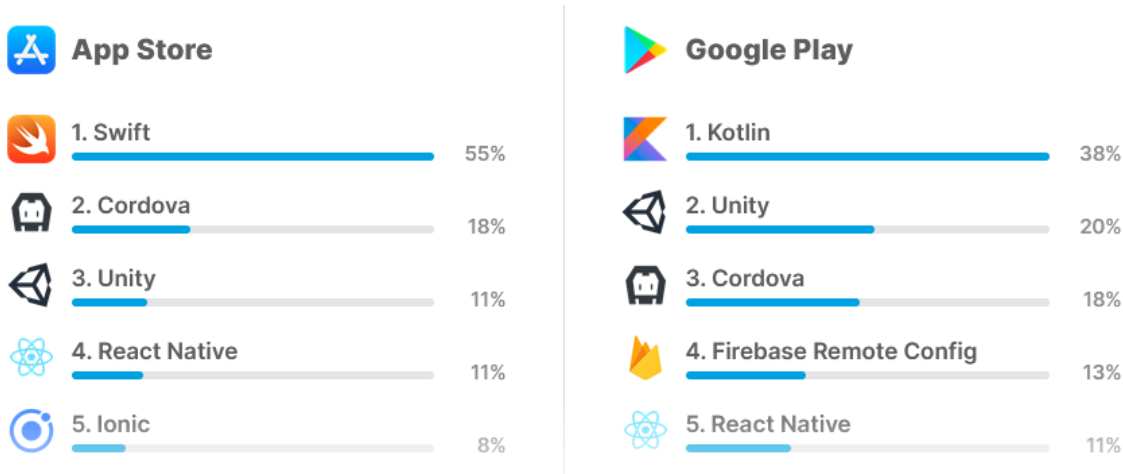


Abbildung 17: Die meistverwendeten SDKs in den App Stores (Quelle: https://appfigures.com/top-sdks/development/all, Stand: 06.04.2022)

Durch die Marktanalyse konnte festgestellt werden, welche Frameworks aktuell auf dem Markt und wie weit sie schon entwickelt sind. Bei der Betrachtung der Tabelle 6 fällt auf,

[15] https://www.google.com/intl/de_at/chrome/

dass sich vor allem die Frameworks Apache Cordova und Capacitor in ihrer Funktionsweise sehr ähneln. Sie sind besser für nicht sehr erfahrene Nutzer geeignet, da der User keine komplizierten Programmierungen mehr selbstständig durchführen muss und vieles automatisch läuft. Electron dagegen ist noch sehr auf die Programmierung konzentriert und deshalb nicht so nutzerfreundlich wie seine Konkurrenten. Die Entscheidung, welches Framework in dieser Arbeit untersucht werden soll, fiel auf Apache Cordova. Capacitor von Ionic ist zwar ein immer besser werdendes Framework, das eine starke Konkurrenz darstellt und voraussichtlich Apache Cordova auf dem Markt überholen wird, jedoch ist Cordova nach jetzigem Stand noch das am meisten verwendete Tool unter den Native Wrappern in den App Stores (vgl. Abbildung 17), weshalb in dieser Arbeit Cordova im Detail besprochen werden soll. Es ermöglicht die Entwicklung von hybriden, mobilen plattformübergreifenden Anwendungen und unterstützt die Offlinefunktionalität von Web Applikationen [44].

4.2.2 Apache Cordova

Apache Cordova (ehemals PhoneGap) ermöglicht es eine Anwendung zu entwickeln, ohne die spezifischen APIs[16] der Betriebssysteme verwenden zu müssen. Es ist ein open-source Framework und wurde von der Firma Nitobi entwickelt. 2011 wurde Nitobi von der Adobe Inc. übernommen, die PhoneGap kurz danach der Apache Software Foundation übertragen haben. Damit wurde auch die Software von PhoneGap zu Apache Cordova umbenannt[17]. Das Framework ermöglicht einer Web App Zugriff auf Hard- und Softwarekomponenten des Gerätes. Da nur eine einmalige Programmierung für mehrere Plattformen nötig ist, werden unter anderem Kosten und Zeit gespart. [44]

Wie in Abbildung 18 zu sehen ist, besteht die Web App aus den Programmiersprachen JavaScript, CSS und HTML. Diese Web App befindet sich innerhalb des Frameworks Cordova und wird von diesem umringt. Die Anwendung wird im "WebView" ausgeführt, welches der Applikation das gesamte User Interface zur Verfügung stellt. [44]

Mit Hilfe der Schnittstelle Cordova Native API hat die Web App Zugriff auf die Cordova Plugins. Diese ermöglichen der Web App die Verwendung der nativen Funktionalitäten des Gerätes, wie zum Beispiel der Kamera, der Geopositionierung oder des Datenspeichers. Die Programmiersprache der Plugins ist dabei von der Plattform abhängig (z.B. Java bei Android). [44]

Apache Cordova bietet nur eine begrenzte Anzahl an Plugins und Funktionalitäten an. Es besteht jedoch die Möglichkeit in Eigenarbeit weitere Plugins zu programmieren oder Plugins von Drittanbietern zu beziehen [5]. Da den Drittanbietern nicht immer vertraut werden kann, muss mit Sicherheitsrisiken gerechnet werden. OS API ist die Programmierschnittstelle zu den Betriebssystemen Android, Apple und Windows.

[16] Programmierschnittstelle

[17] https://cordova.apache.org/announcements/2020/08/14/goodbye-phonegap.html

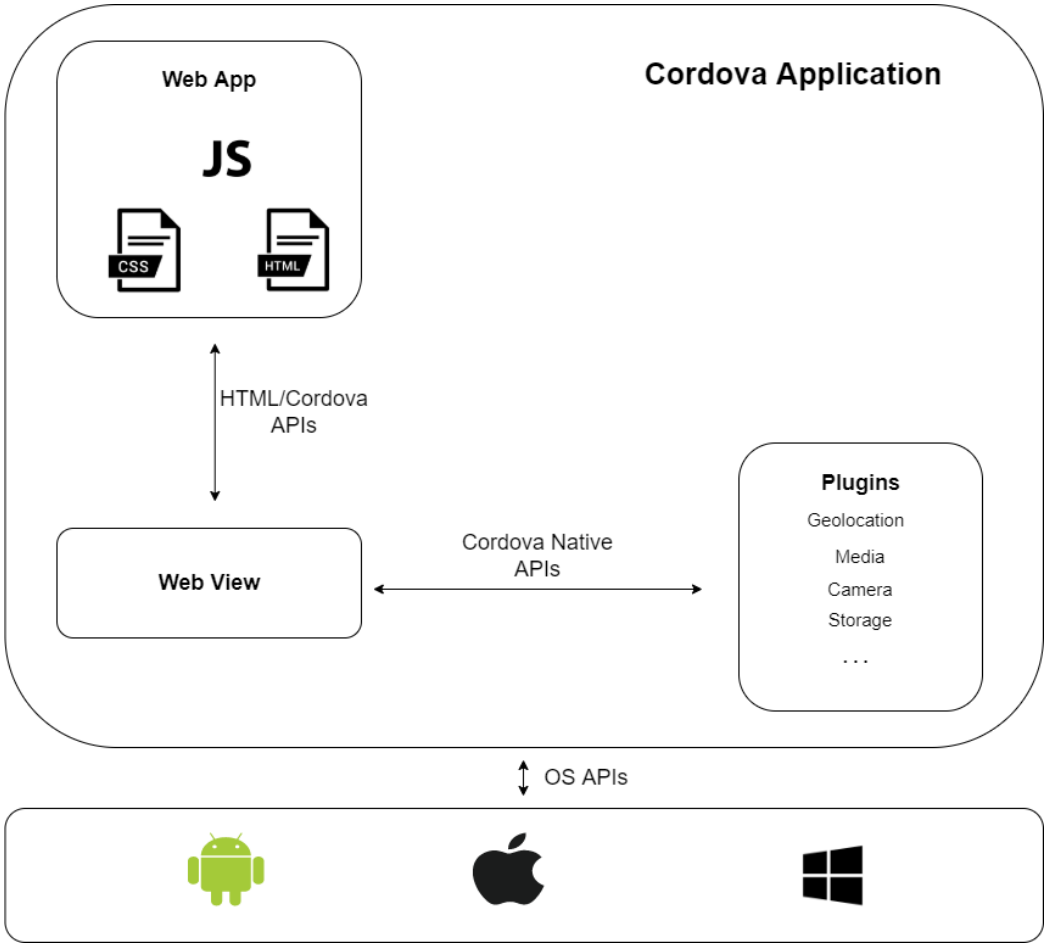


Abbildung 18: Architektur einer Cordova Applikation (Eigene Darstellung)

4.2.3 Einrichtung

Im folgenden Kapitel erfolgt die Erläuterung der Grundeinrichtung von Apache Cordova [44]:

Um Apache Cordova einzurichten, muss zuerst Node.js auf dem Gerät installiert werden. Node.js ist eine plattformübergreifende Open-Source-Javascript Laufzeitumgebung, die JavaScript außerhalb eines Webbrowsers ausführen kann.

Mit Hilfe des Befehls

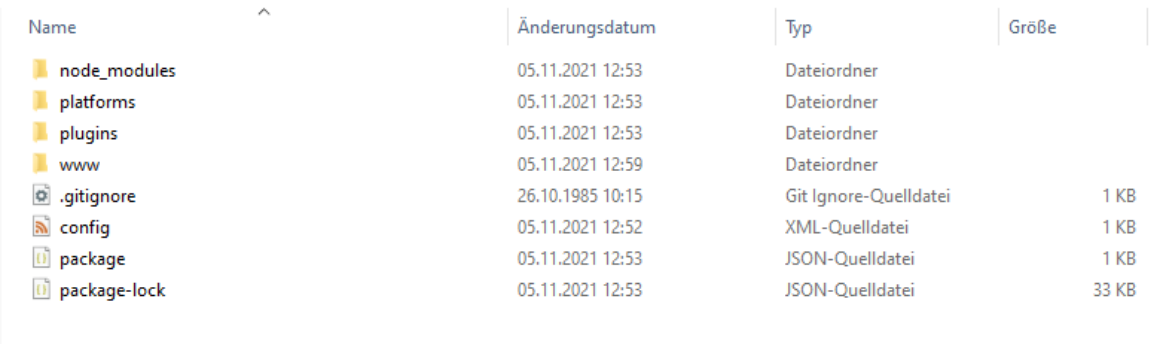
```
C:\>npm install -g cordova
```

wird Cordova auf dem Gerät auf dem Laufwerk C: installiert. "npm" ist ein Paketmanager, der Node.js kostenlos Pakete bereitstellt. "-g" bestimmt, dass Cordova nicht in einem Unterordner, sondern global installiert werden soll.

Mit dem Code

```
$ cordova create hello com.example.hello HelloWorld
```

wird der Pfad festgelegt, wo das Projekt erstellt werden soll. In diesem Projektordner wird anschließend automatisch eine Ordnerstruktur für die weitere Entwicklung der Applikation eingerichtet (vgl. Abbildung 19).



Name	Änderungsdatum	Typ	Größe
node_modules	05.11.2021 12:53	Dateiordner	
platforms	05.11.2021 12:53	Dateiordner	
plugins	05.11.2021 12:53	Dateiordner	
www	05.11.2021 12:59	Dateiordner	
.gitignore	26.10.1985 10:15	Git Ignore-Quelldatei	1 KB
config	05.11.2021 12:52	XML-Quelldatei	1 KB
package	05.11.2021 12:53	JSON-Quelldatei	1 KB
package-lock	05.11.2021 12:53	JSON-Quelldatei	33 KB

Abbildung 19: Ordnerstruktur der Cordova Applikation

Im Ordner "www" befindet sich die Homepage und in dem Ordner platforms befinden sich die Plattformen für, die die Applikation entwickelt werden soll. Je nachdem, welche Plattform gewünscht ist, muss der entsprechende Name hinter folgenden Code gesetzt werden (in diesem Fall "android"):

```
$ cordova platform add android
```

Die Cordova Plugins müssen einzeln installiert werden. Ist zum Beispiel das Kamera Plugin gefordert, muss folgender Code in die Befehlszeile eingegeben werden:

```
$ cordova plugin add cordova-plugin-camera
Fetching plugin "cordova-plugin-camera@~2.1.0" via npm
Installing "cordova-plugin-camera" for android
```

Um zu sehen, welche Plugins schon installiert sind, wird folgender Code verwendet:

```
$ cordova plugin ls
```

Das File "config.xml" (vgl. Abbildung 20) spielt in der Architektur der Cordova Applikation eine wichtige Rolle, da diese Informationen über die Anwendung und wichtige Parameter für die Funktion der App beinhaltet. Dazu zählen der Name, eine Beschreibung der Applikation, der Autor und die Startseite der Applikation. "allow-intent" definiert die URLs, nach denen die Applikation das System fragen darf[18]. Mit "platform" kann entschieden werden, welche Elemente speziell nur für eine Plattform (Android, iOS, Windows) erlaubt werden sollen. [44]

[18] https://cordova.apache.org/docs/en/10.x/config_ref/

```xml
<?xml version='1.0' encoding='utf-8'?>
<widget id="com.example.OLE" version="1.0.0" xmlns="http://www.w3.org/ns/widgets" xmlns:cdv="http://cordova.apache.org/ns/1.0">
    <name>OpenlayersExample</name>
    <description>
        A sample Apache Cordova application that responds to the deviceready event.
    </description>
    <author email="dev@cordova.apache.org" href="http://cordova.io">
        Apache Cordova Team
    </author>
    <content src="index.html" />
    <access origin="*" />
    <allow-intent href="http://*/*" />
    <allow-intent href="https://*/*" />
    <allow-intent href="tel:*" />
    <allow-intent href="sms:*" />
    <allow-intent href="mailto:*" />
    <allow-intent href="geo:*" />
    <platform name="android">
        <allow-intent href="market:*" />
    </platform>
    <platform name="ios">
        <allow-intent href="itms:*" />
        <allow-intent href="itms-apps:*" />
    </platform>
</widget>
```

Abbildung 20: config.xml Datei

4.3 Lokaler Webserver

Die dritte technologische Möglichkeit, ein Web Applikation offlinefähig zu machen, ist die Einrichtung eines lokalen Webservers auf einem mobilen Endgerät. Dabei ist das Betriebssystem "Android" das Einzige, das dies zurzeit möglich macht, da die Einrichtung über das Open-Source Betriebssystem Linux läuft und auf Android Linux basiert. [47]

Am PC besteht die Möglichkeit, lokal einen privaten Webserver einzurichten, über den eine Applikation gestartet werden kann. Ziel ist es, dasselbe auf einem mobilen Endgerät durchzuführen. Dafür wird jedoch eine Android App benötigt, die im Folgenden näher erläutert wird.

Im ersten Teil des Kapitels wird zunächst der Begriff "Webserver" im Detail erläutert. Darauf folgt die Untersuchung der Applikation "Termux", die es möglich macht einen Webserver lokal auf einem mobilen Gerät einzurichten. Zuletzt findet noch eine genaue Beschreibung der Vorgehensweise bei der Einrichtung eines Webservers statt.

4.3.1 Webserver

Wenn eine Internetverbindung vorhanden ist, läuft der Zugriff auf eine Internetseite folgendermaßen ab: Nachdem ein Nutzer eine Anfrage nach einem HTML-Dokument über den Browser gestellt hat, wird diese über das Internet zum Webserver weitergeleitet. Dieser bearbeitet die Anfrage und sendet das HTML-Dokument über das Internet wieder zurück. Der Webserver ist also eine Software, die Daten an Clients überträgt. Webserver werden lokal, in Firmennetzwerken und überwiegend im Internet eingesetzt.[19]

[19] https://developer.mozilla.org/de/docs/Learn/Getting_started_with_the_web/How_the_Web_works

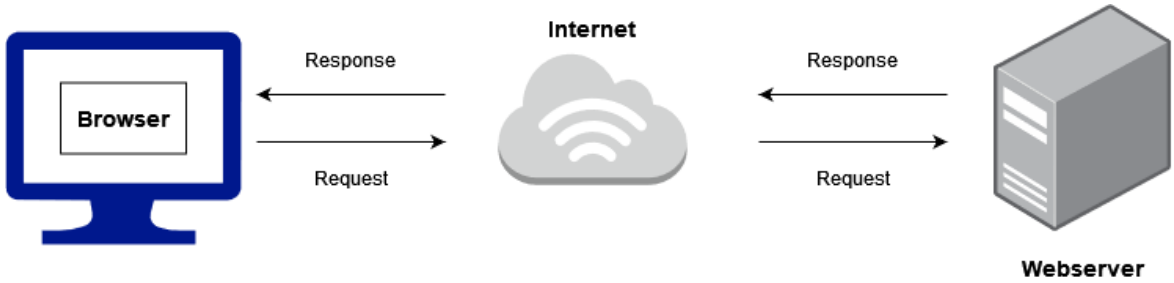


Abbildung 21: Funktion eines Webservers (Eigene Darstellung)

4.3.2 Termux

Wenn keine Internetverbindung vorhanden ist und somit die Kommunikation zwischen Computer und Webserver unterbrochen ist, kann auf dem Gerät lokal ein Webserver eingerichtet werden (vgl. Abbildung 22).

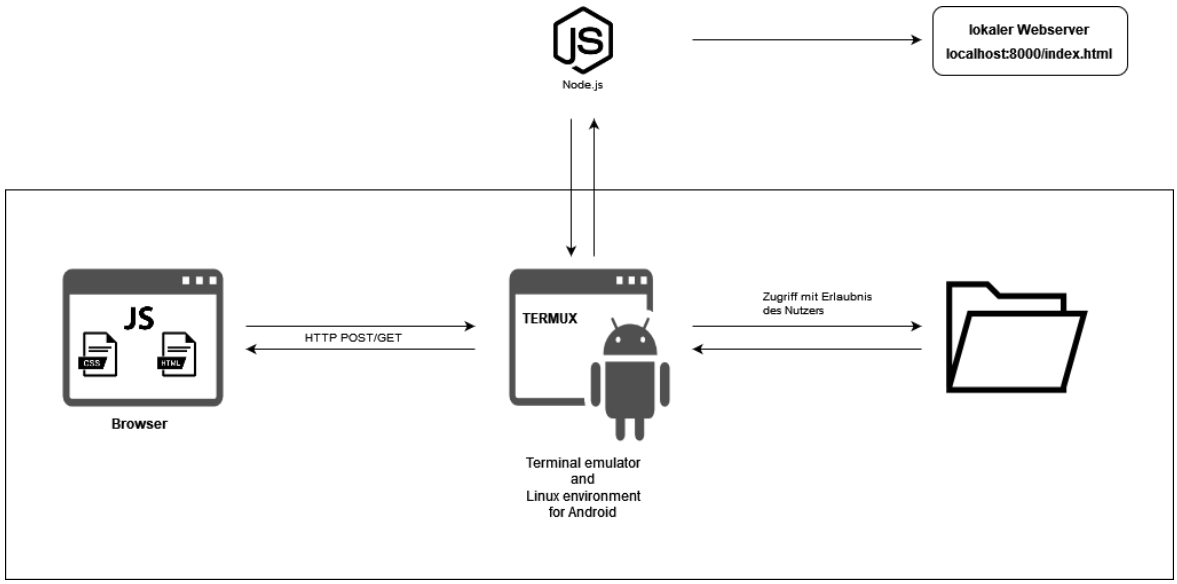


Abbildung 22: Architektur und Funktionalität eines lokalen Webservers (Eigene Darstellung)

Der Nutzer fragt über den Internetbrowser auf dem Gerät eine HTML-Datei an. Normalerweise wird die Anfrage online an den Webserver weitergeleitet und dort verarbeitet. In diesem Fall ist jedoch keine Internetverbindung vorhanden, wodurch der Nutzer keine Antwort auf seine Anfrage bekommen wird. Um die Web Applikation trotzdem zu verwenden, kann ein lokaler Webserver auf dem Gerät eingerichtet werden. Mit einem lokalen Webserver verbindet sich der Browser nicht mit einem netzwerkgebundenen

Webserver. Das bedeutet, dass keine Verbindung zum Internet mehr besteht, sondern nur eine Verbindung zum Gerät besteht.

Für den Webserver wird eine Linux-Umgebung benötigt. Hier kommt das Tool Termux zum Einsatz. Termux richtet einen Terminal-Emulator auf dem Gerät ein, der eine Linux-Umgebung zur Verfügung stellt und über die Linux-Standard-Werkzeuge verfügt. Die Applikation Termux:API ermöglicht damit dem Nutzer den Zugriff auf Hardware-Schnittstellen (Sensoren, Kamera, Telefon, …) des mobilen Geräts. Mit Hilfe von Node.js kann eine Verbindung zu einem lokalen Webserver hergestellt werden (vgl. Abbildung 22). Um die Offlinefunktionalität zu erreichen, müssen die zu verwendenden Daten lokal in einem internen Speicher abgelegt und auf Nachfrage dem Nutzer bereitgestellt werden. Die Möglichkeit der Speicherung der Daten auf dem Gerät ist jedoch nur durch Zustimmung des Nutzers möglich. Damit wird Termux die Erlaubnis erteilt, auf das Dateisystem zuzugreifen. [47]

4.3.3 Einrichtung

Die Einrichtung des lokalen Webservers auf einem mobilen Android Gerät beginnt mit dem Download der Termux App vom Open-Source App Store F-Droid oder von Github. Aufgrund der mangelnden Bereitstellung von aktualisierten Versionen wird der Download aus dem Google Play Store seit 2020 aus Sicherheitsgründen nicht mehr empfohlen. Für eine problemlose Funktion der App ist mindestens die Android Version Nougat (Version 7) empfehlenswert [48].

Die Einrichtung von Termux erfolgt folgendermaßen [48]:

Nach der Installation einer App auf einem mobilen Gerät muss die App gestartet und zuerst aktualisiert werden. Dies wird durch Eingabe folgender Befehle erreicht:

```
apt update
```

```
apt upgrade
```

Um den Speicher einzurichten muss folgender Befehl ausgeführt werden: detailliertere

```
Termux-setup-storage
```

Nach der Zusage für den Zugriff auf das Dateisystem wird ein Verzeichnis im Hauptordner angelegt. Zur Installation des Tools termux-api wird folgender Befehl benötigt:

```
apt install termux-api
```

Für eine vereinfachte weitere Arbeit sollte noch ein Texteditor installiert werden:

```
pkg install nano
```

Installation von Termux Tools:

Die Befehle für die Installation der Tools sind unter *https://wiki.termux.com/wiki/Termux:API* einsehbar. Mit `termux+TAB` lassen sich alle verfügbaren Tools anzeigen:

Abbildung 23: Übersicht der verfügbaren Tools in Termux (Quelle:[48])

Nach der Installation von Termux kann wie am Computer auch über Node‚js der lokale Webserver gestartet werden.[20]

[20] https://medium.com/@huffypiet/how-i-set-up-apache2-web-server-with-termux-on-android-2d7e31aac63e

5 Analyse der Lösungsansätze

Die im letzten Kapitel vorgestellten Lösungsansätze werden im Folgenden im Detail analysiert. Der Vergleich der Technologien basiert auf den Anforderungen, die im Workshop ausgearbeitet wurden (vgl. Kapitel 3.1). Darauf folgt die Gewichtung der Bewertungskriterien, um die Ansätze angemessen beurteilen zu können. Durch eine Gegenüberstellung werden die Anforderungen nach ihrer Wichtigkeit bewertet. Die Gewichtung wird im Anschluss an die Beurteilung an die Bewertung angebracht. Die Beurteilung der Lösungsansätze wird anhand der recherchierten Literatur festgelegt. Die Daten lassen sich aus den Tabellen 7,8,9 und 10 entnehmen. Mit den Ergebnissen werden die Ansätze evaluiert und der am besten bewertete wird im praktischen Teil dieser Arbeit umgesetzt.

5.1 Vergleich der Technologien

Anhand der erarbeiteten Anforderungen werden die drei Technologien näher untersucht und miteinander verglichen. Die Untersuchung ist nach den Überbegriffen Usecase, Datenmanagement, Setup und Funktionalität aufgebaut.

5.1.1 Usecase

In dieser Kategorie wird anhand von vier Kriterien der Anwendungsfall untersucht. Zum Anwendungsfall zählen alle Informationen, die dazu beitragen können, das Ziel der Offlinefunktionalität einer Web Mapping Applikation zu erreichen. Tabelle 7 soll diesbezüglich einen Überblick gewähren.

Anforderungen	PWA	Cordova	Webserver
Erfassungstätigkeiten	Einrichtung möglich	Wenn in der Web App enthalten	Wenn in der Web App enthalten
Lizenz	W3C	Apache 2.0	GPLv3
Offlinefunktionalität	ja	ja	ja
Funktionalität im Intranet	ja	ja	ja

Tabelle 7: Übersicht der Anforderungen bezüglich des Usecases (Eigene Darstellung)

Erfassung und Kartierung

Die erste erarbeitete Anforderung ist die Möglichkeit der Erfassung und Kartierung von Daten (vgl. Tabelle 7). Dieses Kriterium wird für Klienten untersucht, die in der Vermessung tätig sind, da sich die aufzunehmenden Gebiete oft in Gegenden befinden, wo keine oder nur eine geringe Netzwerkverbindung zur Verfügung steht. Die Offlinefunktionalität kann in diesem Fall sehr von Vorteil sein.

Über eine Web Mapping Applikation kann die Karte im Außendienst im Onlinezustand abgerufen und die Messungen online gespeichert werden. Wenn es während der Aufnahmearbeiten zu einer Netzwerktrennung kommt, ist eine Fortsetzung der Arbeit nicht mehr möglich. Mit der Offlinefunktionalität kann die Vermessungsarbeit in diesem Fall jedoch fortgesetzt werden.

Bei Progressive Web Apps ist die Erfassung und Kartierung von Daten möglich. Wie in Kapitel 6.2 beschrieben, werden für die Offlinefunktionalität je eine JavaScript-Datei für die Datenbank sowie eine JavaScript-Datei für die Karte erstellt. Anhand dieser Dateien ist es dem Nutzer der App möglich, Daten, Linien, Polygone und Punkte auf die Karte zu zeichnen und damit Daten zu erfassen. Diese Daten werden direkt in der IndexedDB gespeichert und stehen im Falle einer Netzwerktrennung im Offlinemodus zur Verfügung. Bei der Wiederherstellung der Internetverbindung sind alle neu erstellten Daten noch verfügbar. Wenn mehrere Nutzer in der Web App auf dieselben Daten zugreifen wollen, kann es zu Konflikten kommen. Dieses Thema wird hier nicht weiter untersucht, da es den Rahmen dieser Arbeit sprengen würde.

Bei Apache Cordova werden Web Apps in das Framework eingebunden. Durch die Installation von Plugins erhält die Web App damit den Zugriff auf die nativen Schnittstellen. Eine Web Mapping Applikation bekommt unter anderem dadurch die Möglichkeit, den Positionierungsdienst des Gerätes zu benutzen. Die Kartierung und Datenerfassung mit einer Applikation, die mit Apache Cordova entwickelt wurde, wurde in dieser Arbeit nicht getestet. Aus der Literatur ist jedoch entnehmbar, dass diese Funktion im Offlinezustand möglich ist, wenn die Web App diese vor der Konvertierung schon besaß. [44]

Mit dem lokalen Webserver kann die Applikation offline funktionieren. In diesem Fall ist es kein Tool, das die Offlinefunktionalität möglich macht, sondern der lokale Webserver. Die Web App besitzt somit im Offlinemodus dieselben Funktionen wie vorher. Auch hier sollte die Erfassung und Kartierung von Daten funktionieren.

Lizenz

Die Technologien in den Lösungsansätzen müssen eine Open-Source-Lizenz besitzen.

Bei Progressive Web Apps ist der Service Worker die entscheidende Software, die eine Lizenz besitzt. Es handelt sich hierbei um eine Lizenz des World Wide Web Consortium (W3C)[21], die dem Nutzer die Erlaubnis gibt, unter bestimmten Voraussetzungen die Software kostenlos zu verwenden.

Apache Cordova besitzt die Apache 2.0 Lizenz[22]. Auch hier ist eine kostenlose Verwendung der Software unter vorgegebenen Voraussetzungen möglich.

Die beim lokalen Webserver verwendete Software ist ebenso frei[23].

[21] https://www.w3.org/Consortium/Legal/2015/copyright-software-and-document
[22] https://www.apache.org/licenses/LICENSE-2.0
[23] https://www.gnu.org/licenses/gpl-3.0.de.html

Offlinefunktionalität

Da sich die Masterarbeit mit den Möglichkeiten der Offlinefunktionalität von Web Mapping Apps beschäftigt, ist es unabdingbar, dass alle drei Technologien einen Offlinemodus anbieten.

Die Offlinefunktionalität bei Progressive Web Apps läuft über ein Tool, das in der Web App registriert und installiert wird. Es funktioniert danach wie ein Proxy, indem es Anfragen, die an den Webserver gerichtet sind, selbst verarbeitet (vgl. Kapitel 4.1.2).

Als Hybride Applikation fungiert Apache Cordova als eine Native App. Dadurch ist es ihr möglich, Daten auf dem Gerät zu speichern und offline zu verwenden (vgl. Kapitel 4.2.2).

Der lokale Webserver funktioniert etwas anders. Dieser dient vor allem mobilen Applikationen, die auf dem Endgerät über ein eigenes Netzwerk laufen und somit nicht an eine Internetverbindung gebunden sind (vgl. Kapitel 4.3).

Intranet

Das Intranet ist anders als das Internet nicht öffentlich zugänglich, sondern ein privates Netzwerk, das nur berechtigten Personen einen Zugriff erlaubt. Es wird normalerweise von Unternehmen, Universitäten oder Behörden verwendet.[24]

Aufgrund einer nicht vorhandenen Testmöglichkeit konnten die Lösungsansätze nicht in einem Intranet ausgeführt werden.

5.1.2 Datenmanagement

Die Daten, die im Offlinemodus abgerufen werden, müssen aus einem Cache oder einer Datenbank abgerufen werden. Dementsprechend muss untersucht werden, welche Speichermöglichkeit für eine Web Mapping Applikation vorhanden sind. In der folgenden Tabelle werden die Technologien nach vier Kriterien in der Kategorie Datenmanagement gegenübergestellt.

Anforderungen	PWA	Cordova	Webserver
Speicherung und Lesen von Daten	Cache und Indexed DB	Lokaler Speicher	Lokaler Speicher
Synchronisation	Über Online-Zugang	Über Online-Zugang	Über Online-Zugang
Formate	Vektordaten, Rasterdaten, Bilder, Videos	Vektordaten, Rasterdaten, Bilder, Videos	Vektordaten, Rasterdaten, Bilder, Videos
Fotos (Speicherung, Verlinkung)	möglich	Mit Kamera Plugin	Mit Kamera Plugin

Tabelle 8: Übersicht der Anforderungen bezüglich des Datenmanagements (Eigene Darstellung)

[24] https://wirtschaftslexikon.gabler.de/definition/intranet-38840

Speicherung und Lesen von Daten

Es wird verglichen welche Speicherungsmöglichkeiten ausführbar sind und wie die Speicherung stattfindet. Dem Nutzer soll ermöglicht werden Geodaten und Kartenkacheln vorübergehend zu speichern. Neben der Datenspeicherung ist auch das Lesen der Daten ein wichtiges Kriterium. Die Applikation soll nicht nur Daten speichern, sondern diese offline auch abrufen können.

Progressive Web Apps haben mehrere Speichermöglichkeiten zur Verfügung (vgl. Kapitel 4.1.4). Hier sollen der Cache Speicher und die IndexedDB kurz dargestellt werden. Die IndexedDB besitzt eine große Speicherkapazität. Die Größe des Speichers ist abhängig von der Kapazität des Festplattenspeichers des Gerätes und vom Browser, in dem die Web App gestartet wurde. Vor allem zur Speicherung von Karten und anderen geographischen Daten, wäre dies für Web Mapping Applikationen von Vorteil [43].

Apache Cordova ist für mobile Geräte empfohlen und setzt auf den lokalen Speicher des Gerätes. Zusätzlich lässt sich der Speicher abhängig vom Gerät vergrößern. [44]

Bei der Anwendung des lokalen Webservers läuft die Web Applikation auf dem mobilen Gerät. Auch in diesem Fall werden die Daten lokal auf dem Gerät abgespeichert. [47]

Synchronisation

Bei Wiederverbindung mit dem Netzwerk muss eine Synchronisation der Daten geschehen. Hierbei sollen die Daten, die während der Offlinephase hinzugefügt oder bearbeitet wurden, bei einer Verbindung mit dem Internet mit dem Speicher synchronisiert und aktualisiert werden.

Bei Progressive Web Apps geschieht das mit Hilfe der Background Synchronization API. Diese API ermöglicht Web Applikationen, Aktionen auch im Falle einer Netzwerktrennung durchzuführen. Die vorgenommenen Veränderungen werden anschließend an den Service Worker geschickt. Sobald der Service Worker erfährt, dass wieder eine Verbindung zum Internet besteht, bearbeitet er die Anfrage und aktualisiert damit die Applikation. Die Background Sync API ist jedoch nur in einem sicheren Netzwerk und nur in bestimmten Browsern funktionsfähig. [49]

Beim Framework Apache Cordova läuft der Prozess ähnlich ab. Auch hier werden Veränderungen an der Applikation lokal gespeichert. Wenn es dann wieder zu einer Netzwerkverbindung kommt, werden die Daten synchronisiert. [27]

Die Synchronisation der Daten ist aufgrund der Speichermöglichkeit von Daten auf einem lokalen Speicher möglich, jedoch lässt sich dies explizit anhand von Literatur nicht bestätigen.

Formate

Web Mapping Applikation sollten Bilder, Videos und vor allem Geodaten speichern können. Für die Web Mapping Applikation, die in dieser Masterarbeit erstellt wird, sollte auf jeden Fall das Speichern von Vektor- und Rasterdaten möglich sein.

Aus der Literatur ist entnehmbar, dass die Speicherung von Vektordaten, Rasterdaten, Bilder und Videos bei PWAs (vgl. Kapitel 6.2.2 und Kapitel 6.2.4) [50], bei Apache Cordova [44] und beim lokalen Webserver [47] möglich ist. Abgesehen von Geodaten werden keine weiteren Formate in dieser Arbeit untersucht.

Fotos

Fotos helfen dabei die Ergebnisse in der Kartierung und Datenerfassung übersichtlicher und verständlicher zu machen. Sie können im Außendienst Skizzen und Dokumentationen erweitern bzw. ersetzen. Damit ist es von Vorteil, diese Funktion offline zu benutzen und Fotos abspeichern zu können.

Wenn der Nutzer ein Bild oder Videos in die App laden möchte, genügt es bei den PWAs einen Upload-Button in der HTML-Datei zu definieren [51]. Es konnte nicht ermittelt werden, ob das Foto anschließend auch geographischen Daten zugeordnet werden kann.

Bei Apache Cordova und beim lokalen Webserver ist die Implementierung eines Kamera-Plugin notwendig, um die Kamera des Gerätes zu verwenden. Die Plugins werden dem Nutzer von der Software bereitgestellt und lassen sich damit installieren. Die Bilder werden anschließend lokal auf dem Gerät gespeichert. [44] [47]

5.1.3 Setup

Der Aufwand des Setups bestimmt, ob die Technologie für ein Unternehmen interessant sein könnte. Auch die Funktion des Updates kann entscheiden, ob die Technologie für den Nutzer passend ist.

Anforderungen	PWA	Cordova	Webserver
Aufwand	- Entwicklung einer Web App - Installation von Service Worker und Web App Manifest - Einrichtung der IndexedDB	- Installation zusätzlicher Software notwendig - Abhängig von der Plattform (Android, iOS, Windows)	Download der Termux App und Installation auf dem Gerät
Update	Neustart der Web App	Plugin für ein automatisches Update	Über einen Eingabebefehl

Tabelle 9: Übersicht der Anforderungen bezüglich des Setups (Eigene Darstellung)

Aufwand für die Einrichtung der Offlinefunktionalität

Der für die Konvertierung der bestehenden zu einer offlinefähigen Web App benötigte Aufwand ist ein wichtiges Kriterium für Unternehmen. Es geht darum zu erfahren, wie eine Offlinefähigkeit hergestellt werden kann und wie aufwändig dieser Prozess ist. Dafür sollen der Umfang der Dokumentation, die Soft- und Hardwarevoraussetzungen, die benötigte Arbeitszeit sowie die benötigten Vorkenntnisse näher betrachtet werden. Die Einrichtung der Offlinefunktionalität erfordert unter Umständen sehr gute Programmierkenntnisse, da sich der Vorgang sonst unnötig in die Länge ziehen könnte. Die genaue Beurteilung lässt sich aus Kapitel 5.3 entnehmen.

Die Konvertierung einer Web App hängt davon ab, welche Funktionen diese momentan besitzt und welche sie im Offlinemodus haben soll. Eine Web Mapping Applikation sollte die Fähigkeit haben, geographische Daten zu speichern und zu bearbeiten. Vor allem zur Erfassung und Kartierung sollten bestimmte Funktionen vorhanden sein. Um eine Web Mapping Applikation zu konvertieren, ist es anschließend notwendig zuerst zu untersuchen, welche Datenbank am besten für die Applikation geeignet ist. Der aufwändigste Teil ist nach eingehender Untersuchung die Einrichtung der Datenbank sowie die Bereitstellung einer Offlinekarte in Kachelform (vgl. Kapitel 6.2.2 und Kapitel 6.2.4).

Eine PWA benötigt den Service Worker, das Web App Manifest und einen Datenspeicher. Der Service Worker wird für die Offlinefunktionalität eingerichtet. Das Web App Manifest wird eingebunden, um die Eigenschaften einer Nativen Applikation zu erhalten und damit die Ausführung der App über den Desktop oder Startbildschirm zu ermöglichen. Die IndexedDB wird eingerichtet, damit im Offlinezustand die Daten gespeichert werden können. Für statische Daten wird der Cache des Service Workers verwendet. Eine detaillierte Dokumentation dieses Prozesses folgt in Kapitel 6.2. [12]

Generell wird Cordova über Node.js eingerichtet, was der Web App den Zugriff auf die nativen Schnittstellen des Gerätes ermöglicht. Dafür müssen Plugins installiert werden, deren Einbindungsprozess auf der Webseite des Frameworks näher erläutert wird [44]. Mithilfe eines Plugins werden zum Beispiel die Daten der App mit Erlaubnis des Nutzers lokal auf dem Gerät gespeichert, welche im Offlinezustand wieder abgerufen werden können. Um eine hybride Android Applikation über Cordova zu entwickeln, müssen auf dem Computer zusätzlich die Software Android Studio, Java Development Kit und Gradle installiert werden. Die App lässt sich über einen Emulator am Computer oder auf dem mobilen Gerät starten. [44]

Für die Einrichtung des lokalen Webservers wird die Termux-App auf das Gerät geladen und dort installiert. Über diese kann der lokale Webserver eingerichtet und gestartet werden[25]. Die Web App kann lokal auf dem Gerät laufen. Beim lokalen Webserver ist es ähnlich wie bei den anderen Ansätzen. Wenn die Web App mit dem lokalen Webserver verbunden ist, muss für die Offlinefunktionalität eine Speichermöglichkeit eingerichtet werden. Dies geschieht über die Termux Applikation. [47]

[25] https://medium.com/@huffypiet/how-i-set-up-apache2-web-server-with-termux-on-android-2d7e31aac63e

Update

Um die Sicherheit und die Funktionalität zu sichern ist es notwendig, dass regelmäßig Updates durchgeführt werden.

Bei Progressive Web Apps werden die Apps durch einen Neustart automatisch aktualisiert [38]. Die Applikation wird hierbei neu geladen und die Änderungen werden automatisch angewendet.

Apache Cordova kann entweder manuell aktualisiert werden oder über ein Plugin [44], das vom Hersteller bereitgestellt wird.

Der lokale Webserver lässt sich über einen Befehl aktualisieren, der in die Eingabeaufforderung eingegeben werden muss. Wenn der Nutzer mit der Programmierung nicht vertraut ist, muss das Update vom Entwickler durchgeführt werden. [47]

5.1.4 Funktionalität

Zuletzt werden die Technologien nach ihrer Funktionalität untersucht. Die Entscheidung für eine Art von Applikation hängt davon ab, wie effizient und plattformübergreifend diese ist. Je größer die Zahl der Nutzer, desto größer ist die Nachfrage.

Anforderungen	PWA	Cordova	Webserver
Performance	Gute Performance durch den Abruf von Daten direkt aus der Datenbank	Die Performance hängt vom Browser und dem Gerät ab	Nicht prüfbar
Kompatibilität	Desktop: Edge, Firefox, Safari, Chrome, Opera Mobil: Android, iOS, Windows[26]	Mobil: Android, iOS	Mobil: Android
Geräte	Desktop, mobile Geräte	Desktop, mobile Geräte (Empfohlen)	mobile Geräte
Kamera	Zugriff möglich	Zugriff über Plugin	Zugriff über Plugin

Tabelle 10: Übersicht der Anforderungen bezüglich der Funktionalität (Eigene Darstellung)

[26] https://caniuse.com/serviceworkers (Zugriff: 18.12.21)

Performance

Die Performance der PWA ist abhängig davon, ob offline Daten gespeichert sind. Wenn Daten in der Datenbank oder lokal gespeichert sind, dauert das Laden der Applikation kürzer. [9]

Beim Native Wrapper hängt die Performance stark von der Leistung des Browsers und des Betriebssystems ab, da die Web App über den Browser gerendert wird und über den Web View läuft [52].

Die Leistung des lokalen Webservers ist davon abhängig, wie komplex die verwendete Applikation ist. Je komplexer die Anwendung, desto länger dauert der Aufruf der App [53].

Kompatibilität

Die Technologien sollen vorzugsweise für mehrere Plattformen kompatibel sein. Wenn nur ein JavaScript-Code für mehrere Plattformen notwendig ist, genügt es diesen Code einmal zu programmieren.

Progressive Web Apps haben den Vorteil, dass sie über eine URL erreichbar sind und somit von vielen Geräten aus ausgeführt werden können. PWAs werden seit der erstmaligen Entwicklung von vielen Betriebssystemen unterstützt. Apple hat 2018 mit der Unterstützung begonnen, bietet jedoch noch nicht alle Funktionalitäten an. [12]

Apache Cordova ist ein Framework, mit dem Apps entwickelt werden können. Dementsprechend ist es damit auch möglich iOS Applikationen zu kreieren. Apache Cordova gibt einer Web App die Möglichkeit native Funktionen zu benutzen. Deshalb kann sie über einen App Store heruntergeladen werden. Die Entscheidung, ob die Applikation in den App Store kommt, trifft der Betriebssystemhersteller.

Die Software, die zur Einrichtung eines lokalen Webservers auf einem mobilen Endgerät benötigt wird, funktioniert nur über das Betriebssystem Android.

Geräte

Progressive Web Apps laufen sowohl auf dem Computer als auch auf mobilen Endgeräten [12]. Die Apps, die über Apache Cordova entwickelt werden, können ebenso auf mehreren Geräten installiert werden, jedoch ist die Benutzung auf einem mobilen Gerät zu empfehlen, da die Desktopbenutzung nicht mehr weiter unterstützt wird [44]. Die Einrichtung eines lokalen Webservers lässt sich nur auf einem mobilen Gerät ausführen [27].

Kamera

Die Kamera ist zur Bestandsaufnahme oder zur Kartierung ein wichtiges Feature. Sie dient einer leichteren Dokumentation und um einen Bezug zu Datenerfassungen zu erhalten.

Wenn der Nutzer in einer PWA direkt über die App auf die Kamera zugreifen und damit ein Foto aufnehmen möchte, sind die getUserMedia() API und weitere Befehle notwendig [51].

Beim Native Wrapper erlangt die Web App über diesen Zugriff auf einige Sensoren und Funktionen des Gerätes. In diesem Fall greift der Web View über die Cordova Native API auf das Kamera Plugin zu und erhält damit die Möglichkeit diesen Sensor zu benutzen. Auch

der Positionierungsdienst des Geräts ist im Falle einer Web Mapping Applikation relevant. Durch Eingabe eines bestimmten Codes in die Befehlszeile wird das Plugin der Applikation hinzugefügt und die Web App erlangt den Zugriff auf die einzelnen Funktionalitäten. [44].

Termux App ermöglicht dem Nutzer auf seinem mobilen Gerät einen eigenen Server einzurichten. Den Zugriff der Web App auf die Sensoren gelingt über das Hinzufügen von Plugins in die Termux App. Anschließend ist es dem Nutzer möglich, die installierten Plugins wie beim Native Wrapper zu nutzen. [47]

5.2 Gewichtung der Bewertungskriterien

Wie in Kapitel 3.2.1.1 beschrieben, werden im Folgenden die Kriterien gewichtet. Die Gewichtung wurde nach dem dort beschriebenen Schema durchgeführt. Die Bewertungsergebnisse der einzelnen Kriterien lassen sich aus Anhang III entnehmen.

Prioritätenanalyse		
Anforderungen		**Bewertung**
ANF1	Open-Source Lizenz	10,4%
ANF2	Offlinefunktionalität	10,4%
ANF3	Speichern und Lesen von Daten	9,3%
ANF4	Erfassungstätigkeiten	9,1%
ANF5	Kompatibilität	8,0%
ANF6	Geräte	7,7%
ANF7	Performance	7,1%
ANF8	Aufwand	6,9%
ANF9	unterstützte Formate	6,6%
ANF10	Updaten	5,8%
ANF11	Speichern und Verlinken von Fotos	5,5%
ANF12	Kamerafunktion	5,2%
ANF13	Synchronisation	4,1%
ANF14	Funktionalität im Intranet	3,8%
	Gesamtbewertung	**100%**

Tabelle 11: Ergebnisse der Prioritätenanalyse (Eigene Darstellung)

Wie aus der Tabelle 11 ersichtlich wird, sind die Lizenz, die Offlinefunktionalität, die Datenspeicherung sowie die Erfassungstätigkeiten am wichtigsten. Diese entsprechen den Hauptanforderungen an eine Web Mapping App. Die Nutzung einer Kamera und sonstiger zusätzlicher Features steht bei dieser Prioritätenanalyse im Hintergrund, da sie eine Funktion darstellen, die zwar wünschenswert, jedoch nicht unbedingt notwendig ist. In die Bewertung fließen außer dem subjektiven Empfinden auch die Bewertungen der betreuenden Firma mit ein.

5.3 Beurteilung der Lösungsansätze

Nach dem Vergleich und der Prioritätenanalyse werden in diesem Schritt die drei Technologien beurteilt und evaluiert. Es soll bestimmt werden, welche dieser Technologien am geeignetsten für eine Web Mapping Applikation ist.

In Tabelle 12 sind die Gewichtung der Anforderungen, die Bewertung und die Punkte, die sich aus dem Produkt von Gewichtung und Bewertung ergeben zu sehen. Summiert ergeben die Punkte jeweils die Gesamtbewertung der einzelnen Lösungsansätze. Die Beurteilung basiert auf den ermittelten Daten (vgl. Kapitel 5.1) zu den Technologien.

ANF	Bewertungskriterium	Gewichtung	PWA		Native Wrapper		Lokaler Webserver	
			Bewertung	Punkte	Bewertung	Punkte	Bewertung	Punkte
ANF1	**Open-Source Lizenz**	**10,44%**	**5**	**0,522**	**5**	**0,522**	**5**	**0,522**
ANF2	**Offlinefunktionalität**	**10,44%**	**5**	**0,522**	**5**	**0,522**	**5**	**0,522**
ANF3	**Speichern und Lesen von Daten**	**9,34%**	**5**	**0,467**	**5**	**0,467**	**5**	**0,467**
ANF4	**Erfassungstätigkeiten**	**9,07%**	**10**	**0,450**	**10**	**0,450**	**10**	**0,450**
ANF4-A	Datenerfassung	4,50%	5	0,225	5	0,225	5	0,225
ANF4-B	Kartierung	4,50%	5	0,225	5	0,225	5	0,225
ANF5	**Kompatibilität**	**7,97%**	**5**	**0,398**	**4**	**0,319**	**2**	**0,159**
ANF6	**Geräte**	**7,69%**	**5**	**0,385**	**4**	**0,308**	**3**	**0,231**
ANF7	**Performance***	**7,14%**	**0**	**0,000**	**0**	**0,000**	**0**	**0,000**
ANF7-A	Ladezeit*	5,36%	0	0,000	0	0,000	0	0,000
ANF7-B	Energieverbrauch*	1,79%	0	0,000	0	0,000	0	0,000
ANF8	**Aufwand**	**6,87%**	**21**	**0,289**	**15**	**0,206**	**14**	**0,193**
ANF8-A	Dokumentation	1,47%	4	0,059	3	0,044	2	0,029
ANF8-B	Software	1,31%	5	0,065	3	0,039	3	0,039
ANF8-C	Hardware	1,31%	5	0,065	3	0,039	4	0,052
ANF8-D	Arbeitszeit	1,64%	4	0,065	3	0,049	3	0,049
ANF8-E	Vorkenntnisse	1,14%	3	0,034	3	0,034	2	0,023
ANF9	**unterstützte Formate**	**6,59%**	**5**	**0,330**	**5**	**0,330**	**5**	**0,330**
ANF10	**Updaten**	**5,77%**	**5**	**0,288**	**4**	**0,231**	**3**	**0,173**
ANF11	**Fotos**	**5,49%**	**5**	**0,206**	**5**	**0,206**	**5**	**0,206**
ANF11-A	Speicherung	4,12%	5	0,206	5	0,206	5	0,206
ANF11-B	Verlinkung*	1,37%	0	0,000	0	0,000	0	0,000
ANF12	**Kamerafunktion**	**5,22%**	**5**	**0,261**	**5**	**0,261**	**5**	**0,261**
ANF13	**Synchronisation***	**4,12%**	**0**	**0,000**	**0**	**0,000**	**0**	**0,000**
ANF14	**Funktionalität im Intranet***	**3,85%**	**0**	**0,000**	**0**	**0,000**	**0**	**0,000**
	Gesamt	**100,00%**		4,1		3,8		3,5

Tabelle 12: Beurteilung der Lösungsansätze (Eigene Darstellung)

***Diesbezüglich sind keine Quellen oder Testmöglichkeiten vorhanden. Den Lösungsansätzen wird für diese Kriterien in der Beurteilung eine Punktzahl von 0 gegeben.**

Bei der Beurteilung ergeben sich für die Progressive Web Apps eine Punktzahl von **4,1**, für den Native Wrapper eine Punktzahl von **3,8** und für den lokalen Webserver eine Punktzahl von **3,5**. Die PWAs schneiden im Ganzen bei der Nutzwertanalyse am besten ab.

Die Anforderungen nach einer kostenfreien Lizenz sowie nach der wichtigen Offlinefunktionalität werden von allen Technologien sehr gut erfüllt. Auch das Speichern und Lesen von Daten sowie die Erfassungstätigkeiten werden ermöglicht. Im Hinblick auf die Kompatibilität und die unterstützenden Geräte erreichen der Native Wrapper und der lokale Webserver nicht die Performance der PWA, die mit vielen Betriebssystemen kompatibel ist, da Apache Cordova kein Desktopsystem und nur iOS und Android unterstützt und der lokale Webserver nur auf mobilen Geräten mit Android funktioniert. Der Aufwand ist nach subjektiver Einschätzung beim lokalen Webserver am größten. In allen Ansätzen werden die gefragten Datenformate unterstützt. Das Updaten der Web App ist bei den PWAs am benutzerfreundlichsten. Fotos können gespeichert werden und die Kamerafunktion ist enthalten. [12] [27] [47]

5.4 Evaluation

Im folgenden Kapitel werden die drei Lösungsansätze noch einmal im Gesamtbild evaluiert.

Bezüglich des **Usecase** entsprechen alle Technologien den Anforderungen zur Entwicklung einer Offline Web Mapping Applikation. Beim Workshop wurde als Beispiel eine Applikation erwähnt, mit der Bäume kartiert werden sollen. Dazu zählen Informationen zu den Bäumen sowie Bilder und dessen GPS-Positionen. Desweiteren ist es wichtig, dass die Technologien Open Source und somit kostenlos sind. Dies ist ein entscheidendes Kriterium bei der Auswahl der zu untersuchenden Technologien. Die Offlinefunktionalität ist bei den drei Technologien gegeben.

Beim **Datenmanagement** wird dargestellt, wie die Daten gespeichert und gelesen werden. Während Progressive Web Apps eher auf browserbasierte Speicher setzen [54], konzentrieren sich Apache Cordova und der lokale Webserver vor allem auf den lokalen Speicher [44] [47]. Wenn sich die Applikation im Offlinemodus befindet, muss auf den Speicher zugegriffen werden, um dem Nutzer auch in diesem Zustand Daten zur Verfügung stellen zu können. Wenn diese Daten offline bearbeitet werden, ermöglichen die Tools die Speicherung der bearbeiteten Daten und der anschließenden Synchronisierung bei wiederkehrender Internetverbindung. [43] [44] [47]

Die Eignung als Web Mapping Applikation wird dadurch bewiesen, dass geographische Datenformate wie Rasterdaten und Vektordaten unterstützt und gespeichert werden können. Für die Erfassung von Daten können Fotos von Vorteil sein. Bezüglich des Datenmanagements sind Progressive Web Apps am besten geeignet. Sie besitzen einen browserinternen Speicher, wodurch auch auf den Festplattenspeicher verzichtet werden kann und der Zugriff auf die Daten schneller erfolgt [9]. Von Nachteil kann die Größe des Speicherplatzes sein.

Das **Setup** ist ein entscheidender Faktor beim Vergleich der verschiedenen Technologien. Deren Einrichtung und Möglichkeit zum Hinzufügen von Funktionen sind bei allen etwas unterschiedlich.

Bei Progressive Web Apps müssen im HTML-Dokument die gewünschten Funktionen durch einen Code eingebunden werden. Hier wird eingestellt, wie die Funktionen arbeiten soll, was sie speichern und wie die Applikation aussehen soll (vgl. Kapitel 6.2). Wenn entsprechende Programmierkenntnisse vorhanden sind, lässt sich das leicht umsetzen. Das Aktualisieren der Applikation ist unkompliziert. Durch einfaches Neuladen der Applikation ist die Web App auf dem neuesten Stand [38].

Wenn das Tool bei Apache Cordova und beim lokalen Webserver eingerichtet ist, genügen einfache Befehle, um gewünschte Plugins einzufügen. In Bezug auf das Setup sind das Framework und der lokale Webserver etwas leichter als PWAs zu konfigurieren. Im Gegensatz zu den Progressive Web Apps müssen für verschiedene Funktionen nur Plugins definiert werden [44]. Auch wenn das Setup bei PWAs etwas aufwändiger ist, sind sie in diesem Fall aufgrund der Benutzerfreundlichkeit angenehmer einzurichten.

Bei der **Funktionalität** einer Technologie kommt es vor allem auf Faktoren wie Performance, Kompatibilität, Geräte- und Schnittstellenunterstützung an. Wenn der Datenspeicher gefüllt ist, leisten Progressive Web Apps ein gute Performance. Die Ladezeit der Web Applikation verkürzt sich, da die Funktionen im Browser durchgeführt werden [9]. Apache Cordova hat zwar eine hohe Entwicklungsgeschwindigkeit, die Performance hängt jedoch vom Browser und vom Betriebssystem ab [52]. Der lokale Webserver wurde diesbezüglich nicht untersucht.

Auch in Bezug auf die Kompatibilität ist die PWA den anderen Technologien gegenüber im Vorteil. Eine PWA ist mit den bekanntesten Browsern und Betriebssystemen kompatibel. Dies macht sie zu einer sehr flexiblen Applikation. Die Technologie des lokalen Webservers hingegen ist für ein Unternehmen, das mit seiner Applikation verschiedene Kunden ansprechen möchte, eher ungeeignet. Die Applikation, die einen lokalen Webserver auf dem Gerät ermöglicht, funktioniert nur über Android, weswegen ein Teil der Nutzer nicht auf die Applikation zugreifen könnte.

Bei einer Gesamtbetrachtung der Eigenschaften der einzelnen Technologien lässt sich feststellen, dass jede ihre Vor- und Nachteile besitzt:

	PWA	Apache Cordova	Lokaler Webserver
Vorteile	- Gute Performance - Kompatibilität mit verschiedenen Betriebssystemen - Datenspeicher im Browser - Installation wie eine Native App	- Plugins, die den Zugriff auf Hard- und Software ermöglichen - Zugriff auf native Schnittstellen möglich - Plattformunabhängig	- Plugins, die den Zugriff auf Hard- und Software ermöglichen - Zugriff auf native Schnittstellen möglich
Nachteile	- Ausgiebiger Aufwand bei der Einrichtung der Datenbank	- Mangelnde Unterstützung von Betriebssystemen und Geräten - Aufwendige Konvertierung	- Nicht benutzerfreundlich - Mangelnde Unterstützung von Betriebssystemen und Geräten - Aufwändige Konvertierung

Tabelle 13: Vor- und Nachteile der Lösungsansätze (Eigene Darstellung)

Progressive Web Apps profitieren von der guten Performance [9] und der Kompatibilität mit verschiedenen Betriebssystemen und Browsern. Der Browser ermöglicht dazu dem Nutzer eine Datenspeicherung. Die Installation der Web Applikation auf dem Homescreen lässt die PWA wie eine Native App wirken. [12] Dennoch ist es von Nachteil, dass PWAs einen ausgiebigen Aufwand benötigen, um sie zu entwickeln. Vor allem die Einrichtung der Datenbank stellt eine große Hürde dar.

PWAs eignen sich sehr gut für Web Mapping Applikationen. Für diese Art von Apps ist eine gute Performance gefragt, welche diese Technologie bieten kann [9]. Durch den Speicher im Browser können Offlinedaten dort hinterlegt werden und müssen nicht auf dem Gerät gespeichert werden [42]. Auch GIS-Funktionen wie das Zeichnen von Polygonen, Linien und Punkten sind mit Progressive Web Apps möglich (vgl. Kapitel 6.2.3).

Die Frameworks dienen einer Web Applikation Schnittstellen eines Gerätes bereitzustellen. Mit Apache Cordova erhält die Applikation die Funktionen und das Aussehen einer nativen App, was sie zu einer hybriden App macht. Als hybride Anwendung besitzt sie somit durch Plugins Zugriff auf Hard- und Software des verwendeten Geräts. [4] Von Nachteil ist die mangelnde Unterstützung der einzelnen Betriebssysteme. Im Verlauf dieser Abschlussarbeit hat die Unterstützung von Betriebssystemen durch Apache Cordova abgenommen. Mittlerweile ist das Framework nur noch mit den mobilen Betriebssystemen Android und iOS kompatibel. Windows wird seit der letzten Version nicht mehr unterstützt. Damit gibt es nur noch die Möglichkeit, die Apps über einen Emulator auf dem Desktop zu verwenden. [44]

Für eine Web Mapping Applikation ist diese Technologie geeignet, da sie einen lokalen Speicher besitzt, auf dem Geodaten begrenzt gespeichert werden können. Vor allem lässt sie sich um die Kamerafunktion und einen GPS-Sensor erweitern. [44]

Als dritte Technologie, gibt es den lokalen Webserver. Prinzipiell soll die Verbindung mit dem lokalen Webserver wie beim Computer verlaufen. Diese Technik der Offlinefunktionalität wurde bis jetzt noch nicht eingehend untersucht und stellt eher einen Versuch dar, eine Alternative für bereits bekannte Technologien zu finden. Der Vorteil ist, dass nur die App, die den Webserver bereitstellt, heruntergeladen und auf dem Gerät installiert werden muss. Wenn das Gerät offline ist, kann die App gestartet werden. Das Gerät verbindet sich anschließend mit dem lokalen Webserver und der Nutzer kann auf die Daten zugreifen, die vorher auf dem internen Speicher abgelegt wurden. Auch hier lässt sich eine Applikation einfach mit Plugins erweitern. [47] [48]

Für eine Web Mapping Applikation ist diese Technologie in der Hinsicht geeignet, dass sie offline auf einem mobilen Gerät läuft und Geodaten speichern kann. Für die Technologie muss eine Applikation von einem Drittanbieter heruntergeladen werden, was Sicherheitsrisiken mit sich bringen könnte. Wenn keine Vorkenntnisse in der Arbeit mit Linux-Systemen vorhanden sind, wird es aufgrund einer notwendigen Einarbeitungsphase zu einer längeren Arbeitszeit kommen.

Nach der Gegenüberstellung von Vor- und Nachteilen der Technologien ist der Autor zum Schluss gekommen, eine Progressive Web App umzusetzen. PWAs sind sehr aktuell, nutzerfreundlich und einmal programmiert, vielseitig und auf mehreren Plattformen einsetzbar. Der Nutzer kann mit einer URL auf eine Web Applikation zugreifen und diese anschließend auf seinen Desktop oder Homescreen laden. Der Service Worker, der mit der PWA installiert wird, ermöglicht dem User anschließend die Offlinefunktionalität. Die IndexedDB ist zum Erstellen eine unbequeme Datenbank deren Einrichtung sich jedoch anhand von bestimmten Wrappern vereinfachen lässt (z.B. IDB[27]). Web Mapping Applikationen lassen sich sehr gut zu Progressive Web Apps umsetzen. Im Offlinemodus werden die benötigten Rasterkacheln der Hintergrundkarte in der IndexedDB gespeichert und können dadurch offline verwendet werden. Im Offlinezustand hinzugefügte Geodaten und Attribute werden in der Datenbank des Browsers gespeichert. Sobald wieder eine Internetverbindung hergestellt wurde, werden die alten Daten mit den neuen aktualisiert. (vgl. Kapitel 6.2)

Ziel ist es im nächsten Kapitel die vorhandene Beispielapp in eine offlinefähige PWA umzusetzen. Es soll ermittelt werden, inwiefern eine PWA für eine Web Mapping Applikation geeignet ist und ob diese die erarbeiteten Anforderungen erfüllen kann.

[27] https://github.com/jakearchibald/idb

6 Technologieumsetzung

In diesem Kapitel soll ausführlich darüber berichtet werden, wie einer vorhandenen Web Mapping Applikation eine Offlinefunktionalität verliehen wird. Zuerst soll dem Leser klargemacht werden, was eine Web Mapping Applikation ausmacht. Danach erfolgt die Erläuterung des Aufbaus der App sowie ihrer Funktionen im Onlinezustand. Außerdem werden der Zweck der Anwendung und der Entwicklungsvorgang im Detail beschrieben. Dies soll dem Leser veranschaulichen, inwiefern sich die Applikation nach der Migration zu einer Progressive Web App verändert hat. Um dementsprechend den Vergleich übersichtlicher zu machen, wird dem Leser eine Gegenüberstellung der beiden Vorgehensweisen aufgezeigt. Das Skript zur Beispielapp befindet sich im Anhang I und wird schrittweise erklärt.

Anschließend kommt es zur Umsetzung der Progressive Web App. Zuerst wird erläutert, was im Falle einer Netzwerktrennung passiert und wie das Problem gelöst werden kann. Dafür wird zunächst eine einfache Lösung beschrieben: Das Erscheinen einer selbsterstellten HTML-Seite, die dem App Nutzer auf ansprechende Weise übermitteln soll, dass er sich momentan im Offline-Zustand befindet und deswegen seine Anfrage an den Webserver nicht verarbeitet werden kann. Dies ist jedoch nicht das Ziel, das erreicht werden soll, sondern nur eine Möglichkeit dem App Nutzer die Übermittlung einer schlechten Nachricht zu verschönern.

Der Hauptteil beschreibt, wie man die Technologien, die in Kapitel 4 ausführlich erläutert werden, schließlich umsetzt. Dafür wird zuerst die Einrichtung des wichtigsten Bestandteils einer Progressive Web App, des Service Workers, dargestellt. Hierbei liegt die Konzentration bei der Beschreibung der Registrierung, der darauffolgenden Installation und schlussendlich der Aktivierung des Tools. Diese Schritte sind für eine reibungslose Ausführung des Service Workers essenziell. Damit eine Web App offline funktionsfähig ist, benötigt die Applikation eine geeignete Datenbank, in der Daten gespeichert werden, die bei einer Netzwerktrennung verfügbar sein müssen. Für das Web Mapping ist es notwendig, dass die verwendete Datenbank einen großen Speicher besitzt, um dort Raster- bzw. Vektorkacheln speichern zu können. Die IndexedDB ist eine solche Datenbank. Nach dem Service Worker wird somit die Einrichtung einer Speichermöglichkeit näher erläutert. Dies stellt den kompliziertesten Vorgang bei der Umsetzung einer Progressive Web App dar, weshalb dieser den größten Zeitaufwand im praktischen Teil einnimmt. Nach der Einrichtung der Datenspeicher geht es darum, die Hintergrundkarte der Web Mapping App offline zu speichern. Hier soll es um den Bezug einer Basiskarte und dessen Speicherung gehen. Um es zu einer Web Mapping App zu machen, werden der Applikation noch einige zusätzliche Funktionen hinzugefügt. Dabei handelt es sich um die Möglichkeit der Karte Attribute hinzuzufügen. Dem Nutzer soll es möglich sein, Polygone, Linien oder Punkte einzuzeichnen, die dann in der IndexedDB gespeichert werden. Das Skript zum Hauptteil des Programmcodes wird schrittweise in Anhang II erklärt.

Zuletzt werden die Ergebnisse des Praxisteils zusammengefasst und näher untersucht. In einer Evaluation der Entwicklung der Progressive Web App sollen die Vorgehensweise

sowie eventuelle Probleme diskutiert werden, die während der Programmierung aufgetreten sind. Es sollen mögliche Lösungsmöglichkeiten und Verbesserungen aufgezeigt werden.

6.1 Entwicklung einer Web Mapping Applikation

Das Ziel dieser Arbeit ist es zuerst zu untersuchen, welche technologischen Möglichkeiten vorhanden sind, um eine Web Applikation offlinefähig zu machen. Danach gilt es herauszufinden, welche dieser Technologien am geeignetsten für eine Web Mapping App ist. In diesem Kapitel soll eine beispielhafte Umsetzung einer Progressive Web App vorgestellt werden. Im ersten Schritt muss die Web Mapping App eingerichtet werden. Dies erfolgt mit Hilfe der folgenden Technologien:

JavaScript

Eine plattformübergreifende, objektorientierte Skriptsprache, die vor allem in der Programmierung von Webseiten oder auch in anderen Umgebungen Verwendung findet. Es ist eine Programmiersprache, die sich Jahr für Jahr verändert, wodurch es immer mehr Wege gibt, einen Code zu schreiben. [55]

Cascading Style Sheets (CSS)

CSS ist eine Stylesheet-Sprache, die zusammen mit HTML und JavaScript eine wichtige Rolle in der Webprogrammierung spielt. Mit CSS werden Anweisungen erstellt, wie zum Beispiel Webseiten aussehen sollen. Die Anwendung der Datei findet durch Einbindung in das HTML-Dokument statt. [56]

Hypertext Markup Language (HTML)

HTML ist eine Auszeichnungssprache, die elektronische Dokumente strukturiert. HTML-Dokument beinhalten Hyperlinks, Bilder und andere Inhalte und werden von Browsern dargestellt. Die JavaScript-Dateien sowie die CSS-Datei müssen in das HTML-Dokument eingebunden werden, damit diese verarbeitet und im Webbrowser dargestellt werden können. HTML-Dokumente besitzen bestehen kurz gesagt aus drei Teilen: Dokumenttypdeklaration, HTML-Kopf (Head) und der HTML-Körper (Body). Zuerst gibt es die Dokumenttypdeklaration. Im Head befinden sich außer dem Titel überwiegend Informationen, die nicht im Browser angezeigt werden. Im Body sind die Informationen, die im Browser angezeigt werden. [57]

Openlayers

Für die Web Mapping Applikation wird Openlayers benutzt, eine open-source JavaScript Library, die eine Erstellung von interaktiven Web Maps ermöglicht. Sie wird verwendet, um Karten auf Webseiten einzubinden. Auf der Karte lassen sich Raster-, Vektorkacheln und weitere Geodaten anzeigen. Die Interaktivität zeichnet sich durch die Möglichkeit der Bearbeitung, Hinzufügen und Gestaltung von Geodaten aus. [58]

Neben Openlayers gibt es noch weitere Web Mapping Libraries mit ähnlichen Funktionen:

Library	Typ	URL
Google Maps	Kommerziell	https://developers.google.com/maps/documentation
Leaflet	Open-Source	https://leafletjs.com/
ArcGIS API	Kommerziell	https://developers.arcgis.com/javascript/latest/
Mapbox GL JS	Kommerziell	https://docs.mapbox.com/mapbox-gl-js/api/

Tabelle 14: Web Mapping Libraries (Eigene Darstellung)

Im Gegensatz zu Leaflet sind die Bibliotheken von Google Maps, ArcGIS und Mapbox kommerziell. Da die Software, die in dieser Masterarbeit verwendet wird, kostenfrei sein soll, wäre als geeignete Alternative die Leaflet Library anzusehen. Diese unterscheidet sich durch eine einfachere Codierung, mehr User, aber weniger Funktionen. Openlayers ist für komplexere GIS-Applikationen gedacht, die dementsprechend eine große Anzahl von Formaten unterstützt.

Bei einer Web Mapping App wird die Karte im Kopfteil der HTML-Datei implementiert. Hier müssen die Links zu den benötigten Dateien sowie das Skript der Openlayers Library eingebunden werden. Im Bodyteil wird die Basiskarte gestaltet, d.h. die Position der Karte wird festgelegt, die Ausdehnung im Browser wird eingetragen und sonstige Eigenschaften werden eingerichtet (vgl. Abbildung 25).

Da die App nur entwickelt wurde, um die Technologie der Offline Web Apps zu veranschaulichen, ist die Web Mapping Applikation "Openlayersmap" eine einfache Applikation mit einer Hintergrundkarte. In Abbildung 24 ist der Aufbau der Web Mapping App skizziert. Zu sehen sind eine Basiskarte, die Zoom-Funktionen sowie die Überschrift.

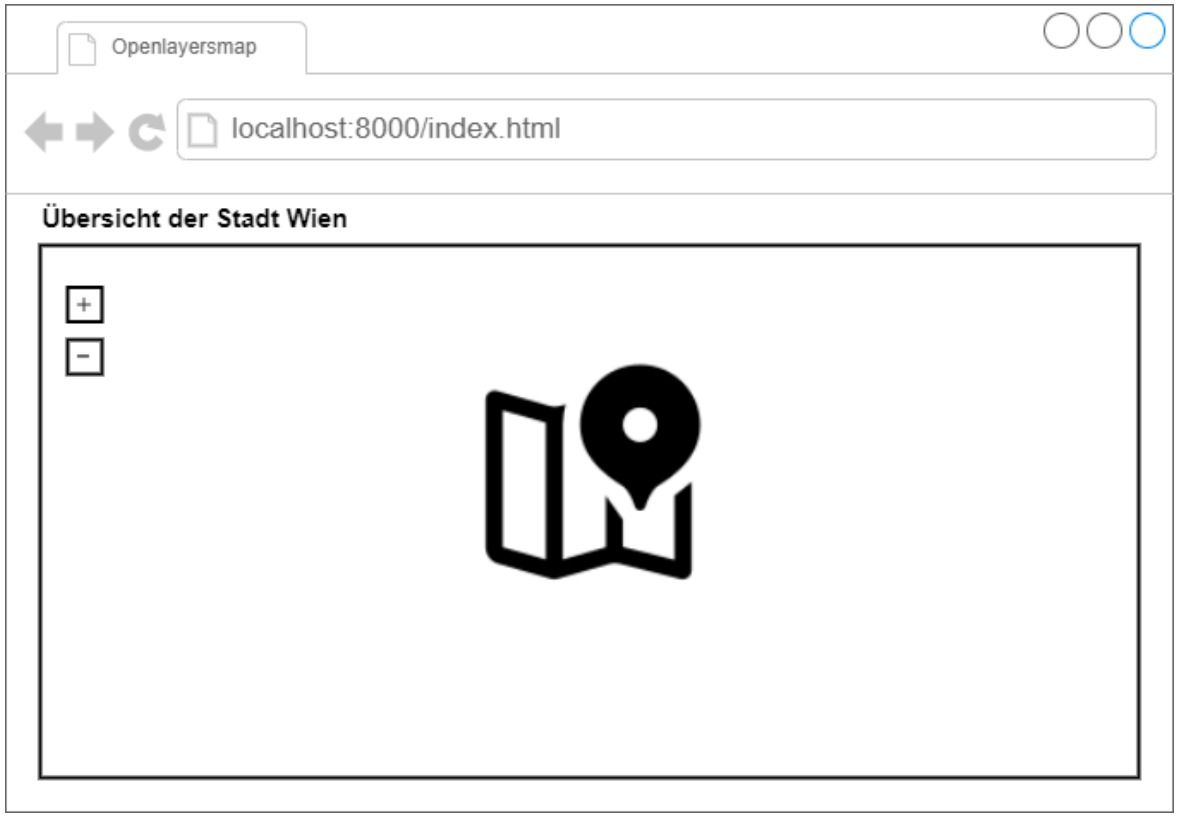


Abbildung 24: Beispielapplikation "Openlayersmap" (Eigene Darstellung)

Mit dem Code (vgl. Abbildung 25) kann eine Basiskarte in die Web Mapping App eingefügt werden. Die Karte wird hierbei über die URL online abgerufen und wird für den verwendeten Zeitraum zur Verfügung gestellt. Zuerst wird der Variable "map" die neue Karte ("new ol.Map") zugeteilt, wobei das ol immer für Openlayers steht. Zur Map gehören ein neuer Layer ("ol.layer.Tile) sowie die Quelle, von der die Karte bezogen wird. Diese Quelle ("ol.source.XYZ") wird verwendet, um Kacheln im XYZ-Format über eine URL bereitzustellen. Die Basiskarte wird über eine URL (http://{a-d}.basemaps.cartocdn.com/dark_all/{z}/{x}/{y}.png) von der Firma Carto bezogen. Die Übermittlung verläuft dabei ohne den Austausch von persönlichen Informationen ("anonymous"). Auf die Einrichtung folgt die Einstellung der Hintergrundkarte. Hier wird anhand von Längengrad und Breitengrad ("ol.proj.fromLonLat") angegeben, welcher Ausschnitt der Weltkarte und mit welcher Zoomstufe er abgebildet werden soll. In diesem Fall handelt es sich um einen Ausschnitt aus der Innenstadt der Stadt Wien.

```
var map = new ol.Map({
    target: 'map',
    layers: [
      new ol.layer.Tile({
        source: new ol.source.XYZ({
          url: 'http://{a-d}.basemaps.cartocdn.com/dark_all/{z}/{x}/{y}.png',
          crossOrigin: 'anonymous'
        })
      })
    ],
    view: new ol.View({
      center: ol.proj.fromLonLat([-0.09,51.505]),
      zoom: 13
    })
  });

```

Abbildung 25: Bezug der Basiskarte für die Web Mapping App

6.2 Migration zu einer Progressive Web App

In folgendem Kapitel geht es nicht nur darum eine Progressive Web App umzusetzen, sondern auch darum, die existierende Applikation um einige Features zu erweitern. Dem Nutzer soll es ermöglicht werden, der Karte Attribute hinzuzufügen, welche dann in der IndexedDB gespeichert werden. Damit können diese im Offlinezustand trotzdem noch abgerufen werden. Durch die Möglichkeit des Hinzufügens von Layern und Attributen (Punkte, Linien, Polygone), stehen somit einige GIS-Funktionen zur Verfügung. Wie im letzten Kapitel beschrieben, macht sie das zu einer Web Mapping Applikation.

Um eine Web Mapping Applikation in eine Progressive Web App umzuwandeln, müssen zuerst der Service Worker und das Web App Manifest sowie eine Datenbank eingerichtet werden, in diesem Fall die IndexedDB. Der Prozess zur Entwicklung einer Progressive Web Mapping App wird in Abbildung 26 dargestellt.

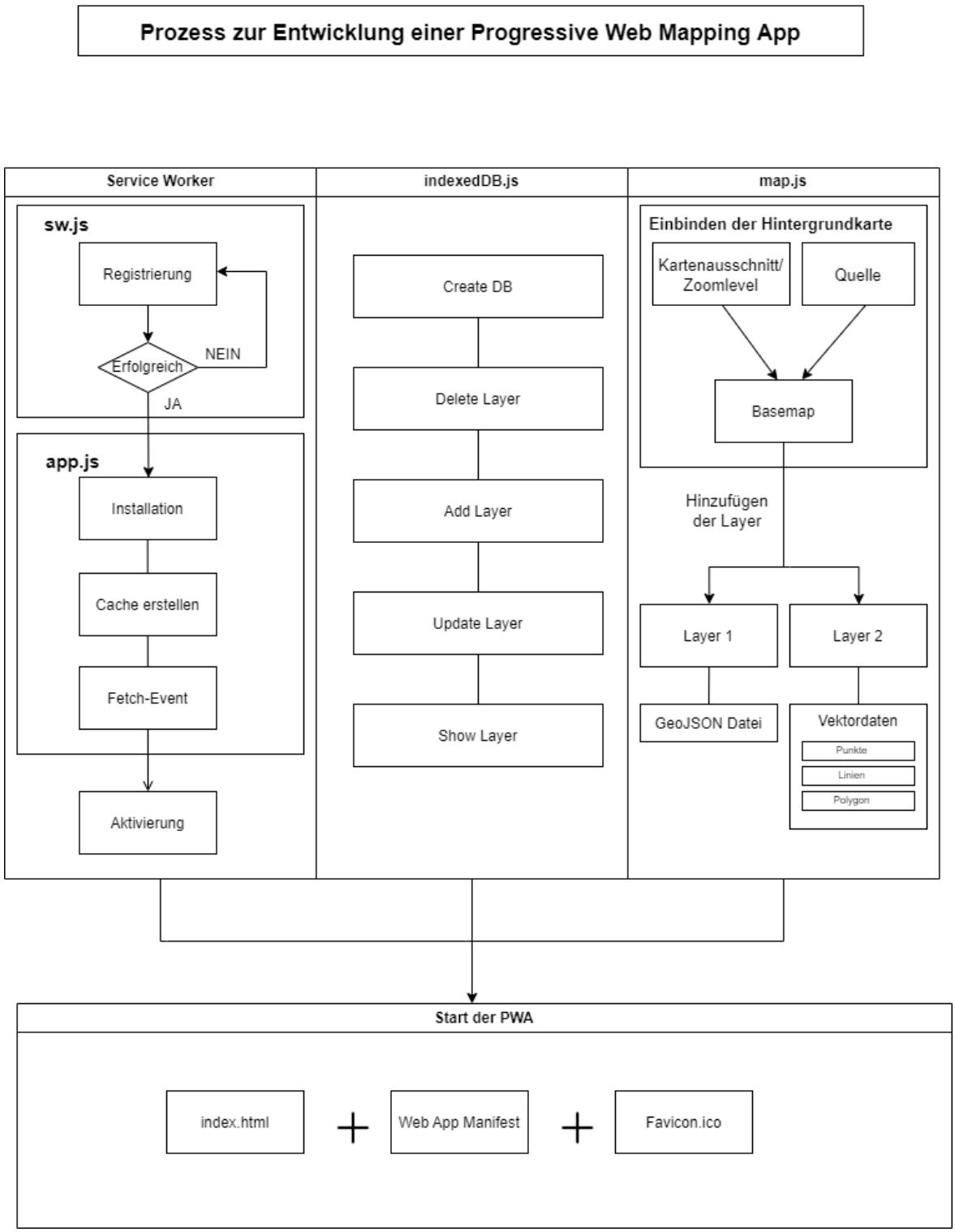


Abbildung 26: Flowchart zur Umsetzung einer Progressive Web App (Eigene Darstellung)

6.2.1 Service Worker – Registrierung, Installation und Aktivierung

Im Folgenden wird wie in Kapitel 4.1.2 beschrieben, der Service Worker eingerichtet:

Dafür muss dieser zuerst im Browser registriert werden. Dafür muss folgender Code in eine separate JavaScript-Datei eingefügt werden (vgl. Abbildung 27). Diese JavaScript-Datei kann einen beliebigen Namen erhalten, jedoch wird sie in dieser Arbeit zum besseren Verständnis "app.js" genannt. Der Zweck dieser Datei ist es, dem Browser den Speicherort des Service Worker (sw.js) zu übergeben. Hierbei wird zuerst kontrolliert, ob der Service Worker vom Browser unterstützt wird. Wenn dies der Fall ist, wird er registriert. Wenn die Registrierung fehlschlägt, muss der Vorgang wiederholt werden. [38]

```
if ('serviceWorker' in navigator) {
    navigator.serviceWorker.register('/sw.js').then(function(reg) {
      // Registrierung erfolgreich
      console.log('Registrierung erfolgreich. Scope ist ' + reg.scope);
    }).catch(function(error) {
      // Registrierung fehlgeschlagen
      console.log('Registrierung fehlgeschlagen mit ' + error);
    });
  };
```

Abbildung 27: Registrierung des Service Workers [59]

Nach einer erfolgreichen Registrierung muss der Service Worker installiert werden (vgl. Abbildung 28). Dies geschieht in der JavaScript-Datei des Service Worker "sw.js". Wenn die Installation fehlschlägt, wird der Vorgang abgebrochen und wiederholt, bis die Installation erfolgt ist. Aktiv wird der Service Worker jedoch erst, wenn die Web Applikation neu geladen wird, da die Applikation durch die Installation des Service Worker nicht automatisch unter dessen Kontrolle steht. Erst wenn alle Browsertabs geschlossen sind, kann die Applikation neug eladen werden und damit den Service Worker aktivieren. Zusammen mit der Installation wird ein Offline-Cache – in diesem Fall "gislayer" - des Browsers eingerichtet, der mit notwendigen Dateien befüllt wird, damit die Applikation offline laufen kann. Die Speicherkapazität des Cache ist jedoch ziemlich begrenzt, weshalb nur die CSS, HTML und JavaScript-Dateien dort gespeichert werden. Weitere Speichermöglichkeiten werden im Verlauf der Arbeit noch näher erläutert (vgl. Kapitel 6.2.4). [38]

```
self.addEventListener("install", function(event) {
  event.waitUntil(
    caches.open("geodata").then(function(cache) {
      return cache.add("/index_offline.html");
    })
  );
});
```

Abbildung 28: Installation des Service Workers [59]

Wenn der Service Worker aktiv ist und die Verbindung zum Webserver getrennt ist, fängt der Service Worker durch das fetch-Event alle Anfragen an den Webserver ab und verarbeitet diese weiter (vgl. Abbildung 29). Dabei gibt es verschiedene Möglichkeiten, wie dem Client geantwortet werden kann. In diesem Fall wird im Offline-Cache "gislayer" nach der Datei "index.html" gesucht, die im letzten Schritt im Offline-Cache gespeichert wurden. Existiert sie, wird sie im Offlinemodus dem Client angezeigt. Existiert sie nicht, kann entweder eine andere Antwort festgelegt werden oder der Client erhält eine Fehleranzeige. [38]

```
self.addEventListener("fetch", function(event) {
  event.respondWith(
    fetch(event.request).catch(function() {
      return caches.match("/index_offline.html");
    })
  );
});
```

Abbildung 29: Fetch Event des Service Workers [59]

6.2.2 Speicherung der Hintergrundkarte

Die Hintergrundkarte der PWA wird aus dem Geodatenviewer der Stadtvermessung Wien bezogen[28]. Hierbei handelt es sich um die Flächen-Mehrzweckkarte (ab sofort FMZK) für den Bereich 35/3 und 35/4[29]. Dieser Bereich umfasst die Innenstadt von Wien (vgl. Abbildung 30). Für die Web Mapping Applikation wird eine georeferenzierte[30] Karte benötigt. Dafür werden die Rasterdaten[31] im GeoTiff-Format mit einer Auflösung von einem Meter heruntergeladen. Zusätzlich werden Text und Geländeschummerung auf der Karte abgebildet.

[28] https://www.wien.gv.at/ma41datenviewer/public/ (Datenquelle: Stadt Wien – data.wien.gv.at)
[29] Blattschnitt: Unterteilung einer Karte in mehrere kleinere Teilkarten
[30] Bei der Georeferenzierung werden einem Geodatensatz raumbezogene Informationen zugewiesen
[31] Raumbezogene Bilddaten, die mit Geoinfomationen verknüpft sind

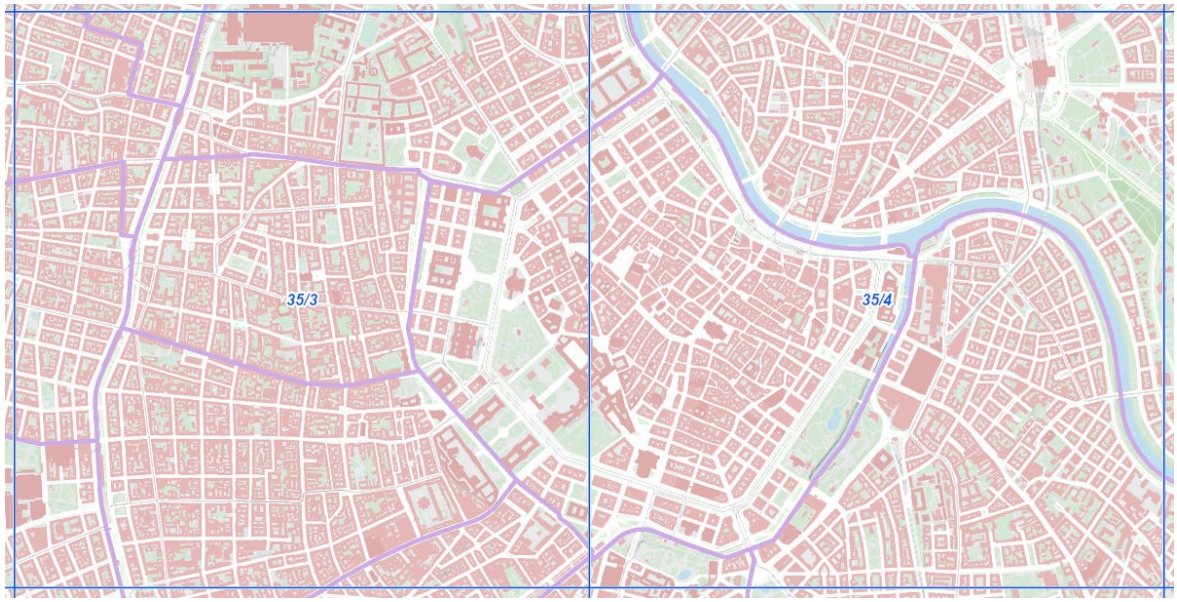


Abbildung 30: Blattschnitte 35/3 und 35/4 der Flächen-Mehrzweckkarte

Im nächsten Schritt werden die bezogenen Blattschnitte mit der Open-Source-Software QGIS weiterverarbeitet. Mit Hilfe eines Werkzeugs werden die Blattschnitte miteinander verschmolzen (vgl. Abbildung 31).

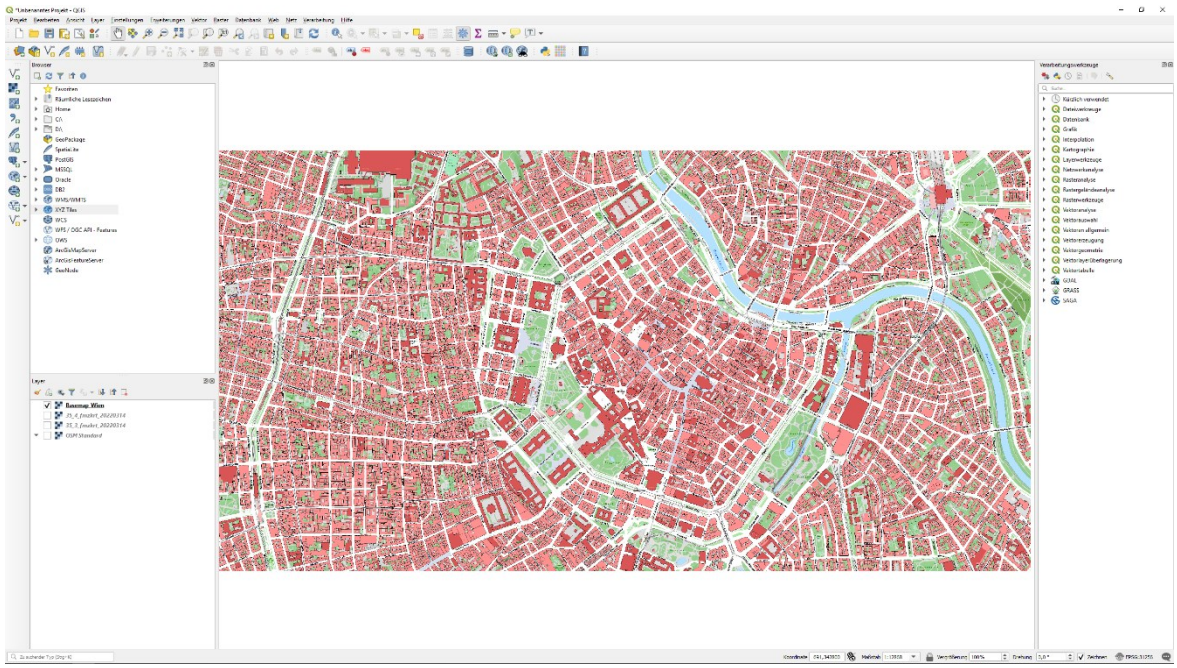


Abbildung 31: Bearbeitung des Rasterbildes in QGIS

Einmal vereint wird das Rasterbild mit Hilfe des Tools "QTiles" in XYZ-Kacheln mit der Pixelgröße 256x256 aufgeteilt (vgl. Abbildung 32). X und Y stehen für die Spalten und Zeilen des Gitters, nach dem die Karte aufgeteilt wurde. Z steht für die Zoomstufe.

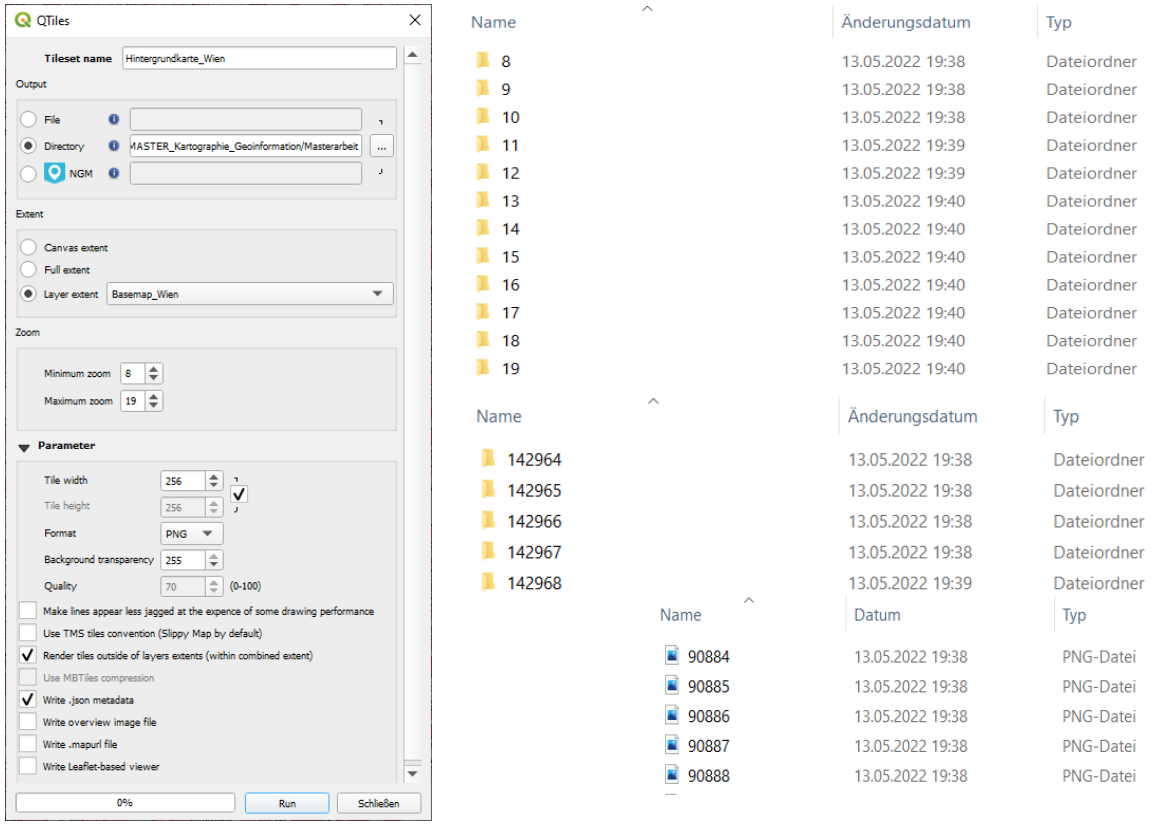


Abbildung 32: Aufteilung der Karte in XYZ-Kacheln mit dem Tool QTiles

Die Hintergrundkarte wird damit lokal auf dem Gerät gespeichert und von dort für die Offlinenutzung der Applikation über den localhost abgerufen (vgl. Abbildung 33). Dies findet über die URL statt. Die URL besteht aus dem Pfad, in dem der localhost gestartet wurde, dessen Portnummer sowie dem Ordner, in dem sich die Rasterkacheln befinden. Im Gegensatz zu einer Web App wie in Kapitel 6.1 dargestellt, findet der Zugriff nicht auf den Internet Webserver sondern auf den lokalen Webserver statt. Die Applikation fragt, je nach gewünschter Zoomstufe, die entsprechenden Kacheln aus dem jeweiligen Ordner ab. Der Anzeigeumfang ist auf den heruntergeladenen Bereich begrenzt.

```
var map = new ol.Map({
    target: 'map',
    layers: [
      new ol.layer.Tile({
        source: new ol.source.XYZ({
          url: 'http://localhost:8000/tiles/{z}/{x}/{y}.png',
          crossOrigin: 'anonymous'
        })
      })
    ],
    view: new ol.View({
      center: ol.proj.fromLonLat([16.37326496188555,48.20842614255267]),
      zoom: 18
    })
  });
```

Abbildung 33: Abruf der Hintergrundkarte

6.2.3 Hinzufügen der Layer und der Funktionen

In dieser PWA soll dem User ermöglicht werden, der Karte Vektordaten[32] hinzuzufügen. Diese werden in der IndexedDB (vgl. Kapitel 6.2.4) gespeichert und stehen offline zur Verfügung. Für die Speicherung werden zwei Layer erstellt (vgl. Abbildung 34). Hierbei wird auch der Style der beiden Layer festgelegt. Darunter zählen Farbe, Form und Linienbreite der Vektordaten. Der erste Layer enthält die GeoJSON-Dateien, der zweite Layer die Punkte, Linien und Polygone.

[32] Raumbezogene Objekte wie Punkte, Linien und Polygone

```
var layer1 = new ol.layer.Vector({
  source: new ol.source.Vector(),
  style: new ol.style.Style({
    fill: new ol.style.Fill({
      color: 'rgba(139, 195, 74, 0.5)'
    }),
    stroke: new ol.style.Stroke({
      color: '#8BC34A',
      width: 3
    }),
    image: new ol.style.Circle({
      radius: 5,
      fill: new ol.style.Fill({
        color: '#8BC34A'
      })
    })
  })
});
map.addLayer(layer1);

var layer2 = new ol.layer.Vector({
  source: new ol.source.Vector(),
  style: new ol.style.Style({
    fill: new ol.style.Fill({
      color: 'rgba(255, 193, 7, 0.7)'
    }),
    stroke: new ol.style.Stroke({
      color: '#00bcd4',
      width: 4
    }),
    image: new ol.style.Circle({
      radius: 7,
      fill: new ol.style.Fill({
        color: '#9c27b0'
      })
    })
  })
});
map.addLayer(layer2);
```

Abbildung 34: Code zum Hinzufügen der Layer (Quelle: [60])

Die Daten, die auf der Karte angezeigt werden, haben ein anderes Referenzsystem[33] als jene, die in der Datenbank gespeichert werden. Das EPSG:3857[34] verwendet dasselbe Koordinatensystem wie eine Karte und dient als Projektion, um die Vektoren auf der Karte abzubilden. Das EPSG:4326 hingegen verwendet ein globales Koordinatensystem und dient als Projektion, um die Vektoren in der Datenbank abzuspeichern. Die Daten werden somit umprojiziert, damit sie auf der Karte abgebildet werden können (vgl. Abbildung 35). Für die IndexedDB müssen die Daten wieder in das ursprüngliche Format konvertiert werden, damit sie von der Datenbank anerkannt und dort gespeichert werden können.

[33] Koordinatensystem im Raum

[34] EPSG: vier oder fünfstellige Codes für Koordinatenreferenzsysteme und weitere geodätische Datensätze

```
64  function featuresToGeoJSON(features){
65    var reader = new ol.format.GeoJSON();
66    var geojson = reader.writeFeatures(features,{
67      featureProjection: 'EPSG:3857',
68      dataProjection: 'EPSG:4326'
69    });
70    return JSON.parse(geojson);
71  }
72
73  function GeoJSONToFeature(geojson){
74    var reader = new ol.format.GeoJSON();
75    var features = reader.readFeatures(geojson,{
76      featureProjection: 'EPSG:3857',
77      dataProjection: 'EPSG:4326'
78    });
79    return features;
80  }
81
```

Abbildung 35: Lesen und Schreiben der GeoJSON-Daten (Quelle: [60])

Für das Einfügen der Daten muss eine Funktion codiert werden, die das Zeichnen der Vektoren ermöglicht. Dafür muss sie zuerst definiert und mit dem Befehl "map.addInteraction(draw)" hinzugefügt werden. Außerdem werden der zugehörige Layer sowie die Datenbank angegeben (vgl. Abbildung 36).

```
101  function draw(type){
102    var draw = new ol.interaction.Draw({
103      source: layer2.getSource(),
104      type: type
105    });
106    map.addInteraction(draw);
107    draw.on('drawend', function (e) {
108      setTimeout(function(){
109        var features = layer2.getSource().getFeatures();
110        var geojson = featuresToGeoJSON(features);
111        database_1.add({
112          type:'updateLayer',
113          param:{
114            layerId:'layer2',
115            geojson:geojson
116          }});
117
118      },100)
119      map.removeInteraction(draw);
120    });
121  }
```

Abbildung 36: Code für die Draw-Funktion (Quelle: [60])

Mit der Funktion "clearAll" wird dem User das Löschen der Daten aus der Datenbank ermöglicht (vgl. Abbildung 37).

```
193   function clearAll() {
194     database_1.add({
195       type:'updateLayer',
196       param:{
197         layerId:'layer1',
198         geojson:{ "type": "FeatureCollection", "features": [] }
199       },
200       callback:function(status){
201         if(status){
202           clearLayer(layer1);
203         }
204     }})
205     database_1.add({
206       type:'updateLayer',
207       param:{
208         layerId:'layer2',
209         geojson:{ "type": "FeatureCollection", "features": [] }
210       },
211       callback:function(status){
212         if(status){
213           clearLayer(layer2);
214         }
215     }})
216   }
```

Abbildung 37: Löschen der Daten aus der IndexedDB (Quelle: [60])

Die Vektordaten lassen sich über die Schaltflächen am oberen rechten Teil der Seite hinzufügen bzw. löschen (vgl. Abbildung 38).

Abbildung 38: Schaltflächen der Funktionen der PWA

Die Farbe, die Größe und die Form der Schaltflächen werden in der HTML-Datei festgelegt (vgl. Abbildung 39).

```
<style>
.map {
    width: 100%;
    height: 1500px;
}

.topRight{
  position: absolute;
  top: 90px;
  right: 0px;
  z-index: 2;
}

button{
  display: block;
  width: 60%;
  height: 40px;
  color: rgb(255, 255, 255);
  background-color: rgb(40, 118, 244);
  border: 1px solid rgb(1, 19, 1);
  border-color: #0e0101;
  cursor: pointer;
  transition-duration: 0,4s;
}

button:hover {
  background-color: rgb(102, 163, 242);
  color: rgba(0, 0, 0, 0.903);
}
.locate {
  top: 4em;
  left: .5em;
}

h1 {
  text-align:center;

}

</style>
```

Abbildung 39: Style der Hintergrundkarte

6.2.4 Einrichtung des Cache Storage und der IndexedDB

Für die Zwischenspeicherung der Daten für den Offlinemodus wird eine Datenbank benötigt. Wie in Kapitel 4.1.4 erwähnt, gibt es verschiedene Speichermöglichkeiten. Darunter zählen der einfache Cache Storage vom Internetbrowser, WebSQL, Web Storage, Local Storage sowie die IndexedDB. Für diese Anwendung werden nur der Cache Storage sowie die IndexedDB verwendet. Web SQL und Web Storage sind veraltet und unpassend für diese Anwendung. Die Verwendung einer lokalen Datenbank stellt eine Alternative zur IndexedDB dar.

Sowohl der Cache Storage als auch die IndexedDB sind Datenspeicher, die sich im Browser befinden. Dadurch muss der Nutzer die Daten, die er für die Web Applikation benötigt, nicht alle auf das Gerät herunterladen. Die Daten werden im Browser gespeichert und von dort wieder aufgerufen. Der Cache Storage ist für statische Dateien besser geeignet, da er nicht viel Speicherplatz zur Verfügung stellen kann und auch nur bestimmte Dateiformate akzeptiert. Die IndexedDB hingegen kann einen großen Speicherplatz bereitstellen und ist damit ideal für eine Web Mapping Applikation, wo die benötigten Daten einen hohen Speicherbedarf haben. Deswegen werden im Cache Storage die HTML,CSS und JavaScript-Dateien gelagert, wohingegen in der IndexedDB die Geodaten gespeichert werden.

Da die IndexedDB umständlich zu konfigurieren ist, wird oft von Wrappern Gebrauch gemacht. Diese sind mit der Datenbank verbunden und sollen die Einrichtung der Datenbank um ein Vielfaches vereinfachen. Darunter zählen IDB[35], Dexie.js[36] und local Forage[37]. Für diese Applikation wird jedoch auf einen Wrapper verzichtet, da die Umsetzung einfach gehalten und nicht zusätzliche Software verwendet werden soll.

Wie in Kapitel 6.2.1 zu sehen ist, wird der Cache gleichzeitig mit dem Service Worker installiert. Hier wird eingetragen, welche Daten im Cache gespeichert werden sollen. Da schon während der Entwicklung der Applikation festgelegt wird, welche Daten im Cache gespeichert werden sollen, ist dieser Speicherplatz besser für Daten geeignet, die nicht mehr geändert werden müssen. In der Entwicklungsumgebung vom Browser kann man feststellen, welche Daten sich aktuell im Cache befinden (vgl. Abbildung 40).

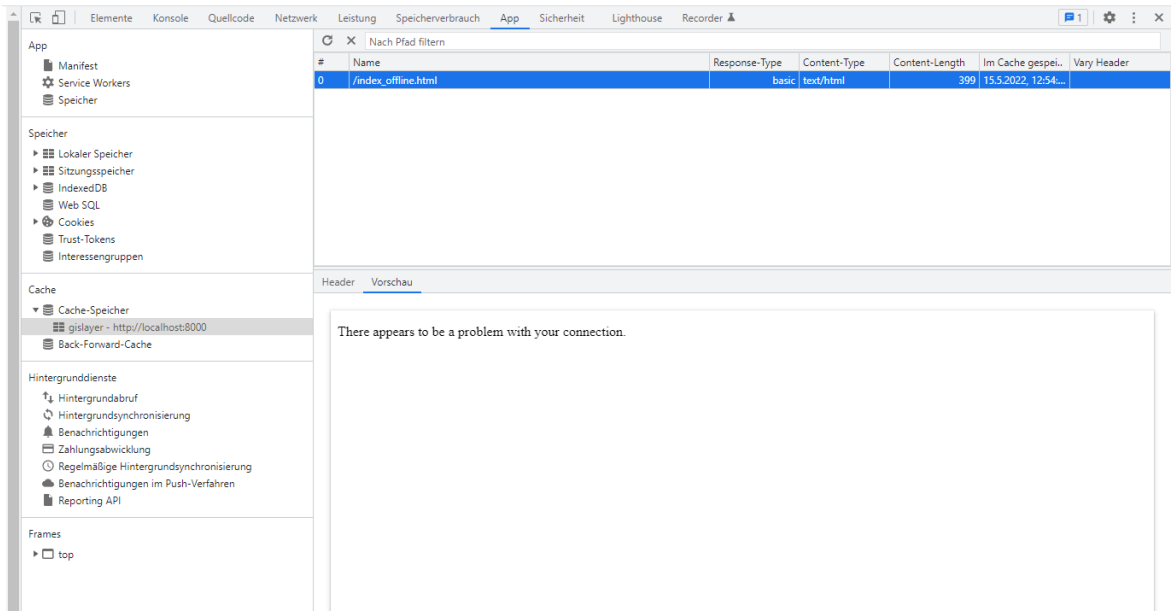


Abbildung 40: Inhalt des Cache Speichers

Eine Aktualisierung muss in diesem Fall manuell stattfinden. Im Offlinemodus wird vom Service Worker geprüft, ob sich die angefragte Datei im Cache befindet. Wenn dies zutrifft,

[35] https://github.com/jakearchibald/idb
[36] https://dexie.org/
[37] https://github.com/localForage/localForage

wird dem Nutzer die Datei angezeigt. Wenn der Service Worker nicht registriert und aktiviert wird, wird dem User eine alternative Webseite dargestellt, die ihn auf den Offlinezustand der Web Applikation aufmerksam macht (vgl. Abbildung 41).

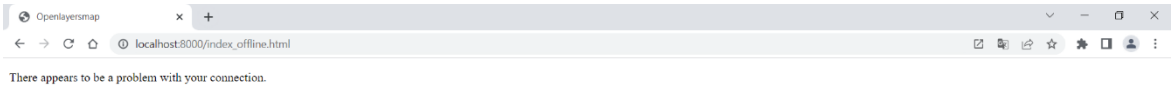


Abbildung 41: Benachrichtigung des Offinezustands

Die IndexedDB wird so erstellt, dass sie global bei allen Browsern funktioniert. Dies erfolgt durch den Code in Abbildung 42.

```
window.indexedDB = window.indexedDB || window.mozIndexedDB || window.webkitIndexedDB || window.msIndexedDB;
window.IDBTransaction = window.IDBTransaction || window.webkitIDBTransaction || window.msIDBTransaction;
window.IDBKeyRange = window.IDBKeyRange || window.webkitIDBKeyRange || window.msIDBKeyRange;

class IDBGeoJSON {
  constructor(params){
    this.db_name = params.db_name;
    this.localStorageName = this.db_name+'-indexedDB-version';
    this.round=parseInt(params.round) || 5;
    this.jobs=[];
    this.add({
      type:'createDB',
      prop:{
        db_name:this.db_name
      }
    })
  }

```

Abbildung 42: Definition der IndexedDB (Quelle: [60])

Neben dem Namen wird auch die Version der Datenbank festgelegt. Diese nimmt bei jeder Veränderung der Datenstruktur die nächsthöhere Versionsnummer an. Die Datenstruktur wird durch jede Transaktion, die vom User durchgeführt wird, geändert (vgl. Abbildung 43).

```
19  setVersion(version){
20    localStorage.setItem(this.localStorageName, version);
21  }
22
23  getVersion(){
24    var version = localStorage.getItem(this.localStorageName);
25    if (version == null || version == '') {
26      localStorage.setItem(this.localStorageName, 1);
27    }
28    version = localStorage.getItem(this.localStorageName);
29    return parseInt(version,10);
30  }
31
32  createDB(callback){
33    var self = this;
34    var request = indexedDB.open(this.db_name);
35    request.onerror = function(event) {
36      console.log('indexedDB Error Code:'+event.target.errorCode);
37      callback(false)
38    };
```

Abbildung 43: Codeabschnitt zur Festlegung der Datenbankversion (Quelle: [60])

Mit Hilfe dieser Methode (vgl. Abbildung 44) wird verhindert, dass sich Transaktionen mit einer ähnlichen Versionsnummer überlappen. Mit der switch Anweisung werden die Transaktionen nacheinander durchgeführt, wodurch verhindert wird, dass die Zweite beginnt, bevor die Erste abgeschlossen ist. Wenn der Wert der case Klausel dem Ausdruck ("job.type) der switch Anweisung entspricht, wird diese durchgeführt. Break bricht die Ausführung ab.

```
164 add(job){
165   this.jobs.push(job);
166   if(this.jobs.length==1){
167     this.next();
168   }
169 }
170
171 next(){
172   var self = this;
173   if(this.jobs.length>0){
174     const job = this.jobs[0];
175     switch(job.type){
176       case 'createDB':{
177         this.createDB((status)=>{
178           if(job.callback!==undefined){
179             job.callback(status);
180           }
181           self.jobs.splice(0,1);
182           self.next();
183         })
184         break;
185       }
```

Abbildung 44: Code zur Verteilung der Transaktionen (Quelle: [60])

Durch den Code (vgl. Abbildung 45) wird die neue Datenbank benannt und der Layer 2, in dem die GeoJSON-Dateien gespeichert sind, wird erstellt.

```
232  var database_1 = new IDBGeoJSON({
233    db_name:'gislayer'
234  });
235
236  database_1.add({
237    type:'addLayer',
238    param:{
239      layerId:'layer2',
240      geojson:{ "type": "FeatureCollection", "features": [] }
241    }
242  });
243
244  database_1.add({
245    type:'showLayer',
246    callback:(layers)=>{
247      layers.map((layer)=>{
248        var features = GeoJSONToFeature(layer.geojson);
249        addFeaturesToLayer(window[layer.layerid],features);
250        zoomToLayer(window[layer.layerid]);
251      });
252    }
253  });
254
```

Abbildung 45: Erstellung von Layer2 (Quelle: [60])

In der Entwicklungsumgebung des Browsers ist nach der Einrichtung der IndexedDB die Datenbank mit den zwei Layern sichtbar (vgl. Abbildung 46). Wenn eine GeoJSON-Datei oder Vektordaten hinzugefügt wurden, werden diese in der IndexedDB gespeichert. Unter Layer 2 sind die Eigenschaften der gezeichneten Geometrien einschließlich der Koordinaten der einzelnen Punkte einsehbar.

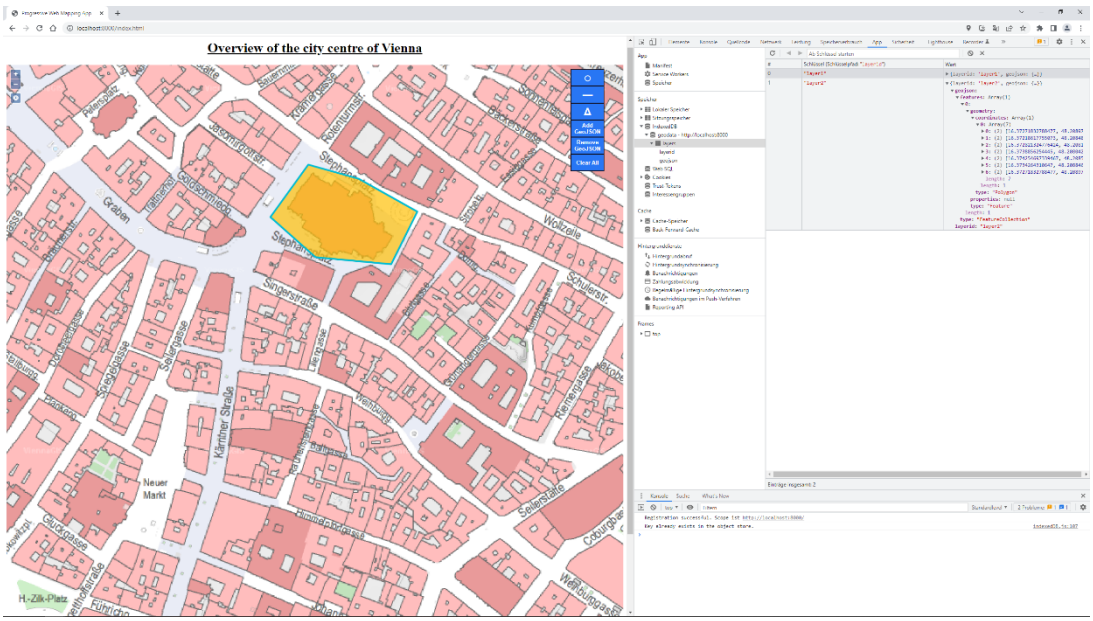


Abbildung 46: IndexedDB im Browser

6.2.5 Start der Applikation

Um die Progressive Web App starten zu können, muss zuerst eine sichere Umgebung eingerichtet werden. Eine PWA funktioniert nur über HTTPS [38]. Da es sich in diesem Fall nur um eine Beispielapp handelt, die nicht über das Internet läuft, wird hier ein lokaler Testserver ("localhost") auf dem Computer eingerichtet.

Um den Localhost einzurichten, muss zuerst, wenn noch nicht bereits erfolgt, die Programmiersprache Python installiert werden. Dafür muss die Software von der offiziellen Homepage heruntergeladen werden. Nun kann über die Eingabeaufforderung mit python -v, python3 -v oder py -v getestet werden, ob die Software installiert ist. Mit der Herausgabe der Versionsnummer durch den Computer wird dies bestätigt. Über die Eingabeaufforderung erfolgt anschließend die Einrichtung des Testservers (vgl. Abbildung 47). Es beginnt mit dem Zugriff auf den Dateipfad. Dafür muss das Laufwerk eingegeben werden, auf dem sich die HTML-Datei der Applikation befindet und anschließend über "cd" der genaue Dateipfad eingefügt werden. Durch die Eingabe von "python -m http.server" oder "py -3 -m http.server", kann der Server gestartet werden. Da der Server lokal auf dem Gerät läuft, befindet sich die Anwendung in einer sicheren Umgebung. Durch Schließen der Eingabeaufforderung lässt sich der Testserver stoppen. [61]

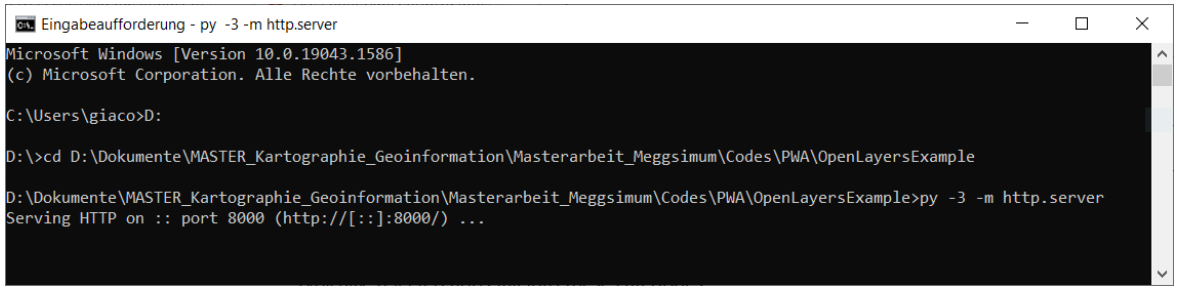


Abbildung 47: Einrichtung des lokalen Webservers

Wenn der Server gestartet wurde, kann über den Internetbrowser auf die Progressive Web App zugegriffen werden. Dafür wird in die Eingabezeile in diesem Fall *"http://localhost:8000/index.html"* eingegeben. 8000 entspricht dabei der Portnummer und index.html ist die Datei, mit der die Applikation gestartet werden kann. Die andere Möglichkeit, um auf die PWA zuzugreifen, besteht über das Icon auf dem Homescreen. Die Applikation kann beim ersten Besuch wie eine Native App auf dem Homescreen installiert und damit in Zukunft einfacher und schneller gestartet werden (vgl. Abbildung 48).

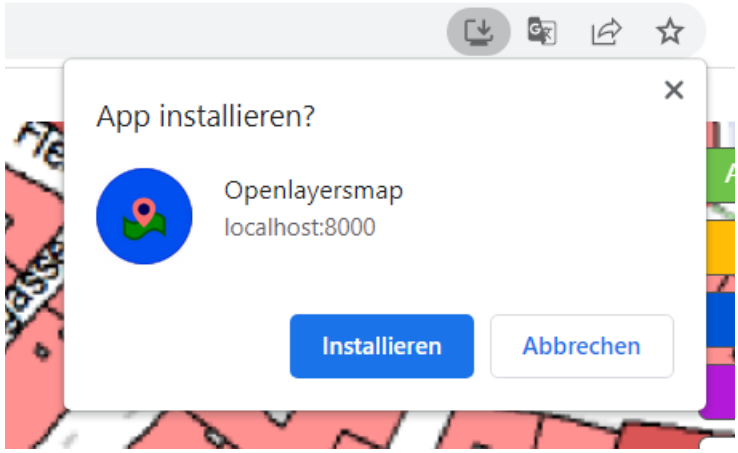


Abbildung 48: Pop-Up für die Installation der PWA auf dem Desktop

Nach der Installation der PWA erscheint auf dem Desktop ein Icon, über das die Applikation ab sofort gestartet werden kann (vgl. Abbildung 49).

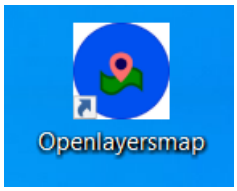


Abbildung 49: Icon über das die PWA gestartet wird

Nach dem Starten ist die App bereit zur Benutzung (vgl. Abbildung 50).

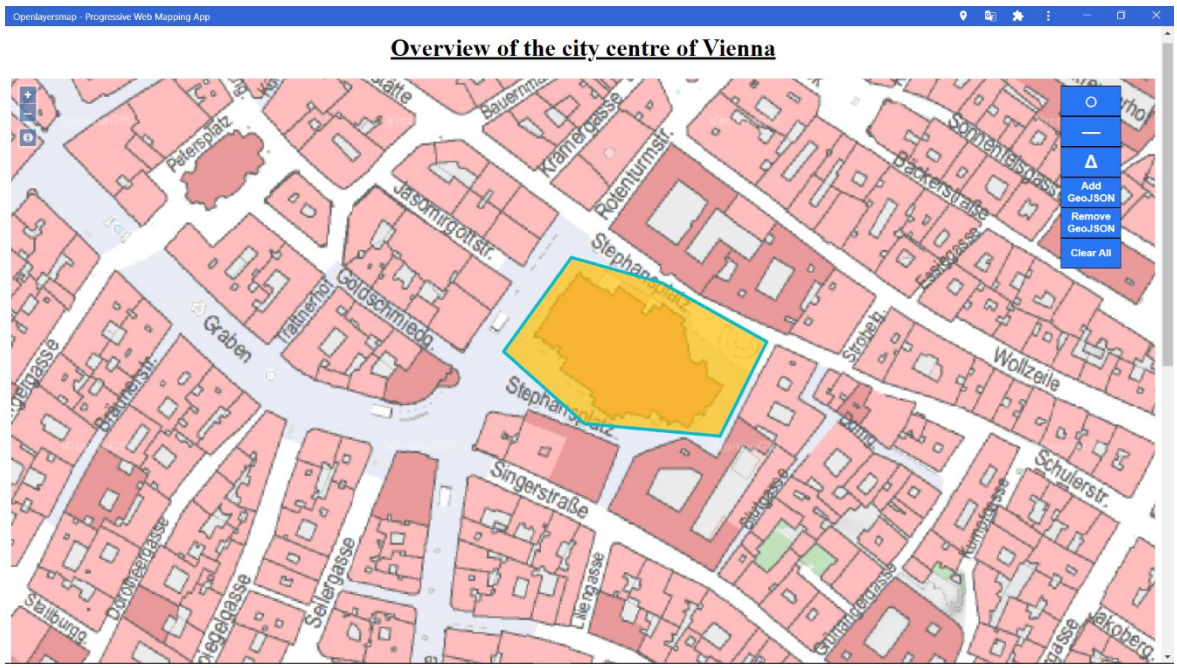


Abbildung 50: Progressive Web Mapping App

6.3 Evaluation der Entwicklung der Progressive Web App

Im folgenden Kapitel wird die Entwicklungsarbeit evaluiert, die für die Migration zu einer Progressive Web App nötig ist. Der Fokus liegt dabei auf der Bewertung des Arbeitsaufwands und der Darlegung der aufgetretenen Schwierigkeiten. Dem Leser soll aufgezeigt werden, ob bei der Umsetzung der PWA die Kosten-Nutzen-Relation stimmt.

Von den Anforderungen an eine Offline Web Mapping Applikation, die zu Beginn der Masterarbeit erarbeitet wurden, konnten die wichtigsten Kriterien umgesetzt werden. Darunter zählen die Offlinefunktionalität, die Open-source Lizenz, die Datenerfassung, das Speichern und Lesen von Daten sowie die Kompatibilität und die Geräteunterstützung. Der Aufwand der Konvertierung und der Einrichtung der Applikation waren aufgrund der ungenauen Dokumentation hoch. Der Aufwand ist jedoch abhängig von weiteren Faktoren, weshalb dieser vom Entwickler abhängt. Des Weiteren wurde die Updatefunktion umgesetzt, die es dem Nutzer einfach macht, die Applikation zu aktualisieren. Wunschkriterien wie die Verfügbarkeit einer Kamera und das Speichern von Fotos, wurden aufgrund ihrer geringen Wichtigkeit nicht umgesetzt. Die Funktionen, die sich auf das Intranet beziehen, wurden nicht getestet.

Hinsichtlich der Arbeitszeit unterscheiden sich die Entwicklung einer Web App und einer Progressive Web App sehr. Die Programmierung der Beispielapp hat weniger Zeit in Anspruch genommen als die Migration der Applikation zu einer PWA. Da diese eine Offlinefunktionalität anbieten muss, ist es notwendig, dass der Service Worker, die Web App Manifest Datei und eine Datenbank hinzugefügt werden [12]. Aufgrund der ausführlichen Dokumentation ließ sich der Service Worker gut einrichten. Für ein komplexeres Beispiel ist die Dokumentation jedoch nicht detailliert genug. Die Bearbeitung und Speicherung der Hintergrundkarte konnte nach Einarbeitung in die Materie durchgeführt werden. Wie in Kapitel 6.2.4 erläutert, ist die Einrichtung der Datenbank in diesem Fall der aufwändigste Prozess. Wenn keine Erfahrung mit der Programmierung einer asynchronen Datenbank vorhanden ist, nimmt die Entwicklung mehr Zeit in Anspruch und ist aufwändiger. Dies war auch der Teil der Arbeit, bei dem die meisten Schwierigkeiten aufgekommen sind.

Der Aufwand eine vorhandene Web App zu einer PWA zu konvertieren, hängt von den Vorkenntnissen ab, die der Entwickler besitzt. Nach einer Einarbeitungsphase lassen sich PWAs mit ihren Grundfunktionen aufgrund der ausführlichen Dokumentation einwandfrei konfigurieren. Wenn das Anwendungskonzept jedoch komplizierter wird, ist die Dokumentation zu oberflächlich. Die IndexedDB gilt als benutzerunfreundlich, weswegen Wrapper benutzt werden, die die Programmierung vereinfachen sollen. Auch der Service Worker lässt sich durch bestimmte Werkzeuge einfacher konfigurieren.

Zusammenfassend kann die Innovation von PWAs mit der Offlinefunktionalität, dem Speichern von großen Datenmengen im Browser und der Installierbarkeit festgestellt werden (vgl. Tabelle 15).

Vergleich von Web Mapping Apps und PWAs:

Eigenschaften	Web App	PWA
Offlinefunktionalität	x	✓
Speichern von großen Datenmengen im Browser	x	✓
Installierbar	x	✓
Datenerfassung	✓	✓
Gute Performance	✓	✓

Tabelle 15: Vergleich von Web Mapping Apps und PWAs (Eigene Darstellung)

Die Daten für die Progressive Web App sind auf Github unter folgendem Link verfügbar:

https://github.com/GiaK0/Progressive-Web-Mapping-App

7 Zusammenfassung

7.1 Fazit

In Kooperation mit dem Büro für Geoinformatik Meggsimum wurde in dieser Arbeit die Offlinefunktionalität von Web Apps näher betrachtet. Der Fokus lag dabei auf Web Mapping Applikationen, die dem User, zusätzlich zu den standardmäßigen Funktionen einer Web Map wie die Zoom Funktion und die Betrachtung von verschiedenen Kartenarten (MZK, Orthophoto, Oberflächenmodell, Geländemodell) noch GIS-Funktionen (Zeichnen, Koordinatenbestimmung, Hoch- und Herunterladen von Geodaten) bieten. Am Anfang wurden im Rahmen eines Workshops Anforderungen erarbeitet, welche die Firma und mögliche Kunden an eine Offline Web Mapping App haben. Diese wurden unterteilt in Usecase, Datenmanagement, Setup und Funktionalität. Zum allgemeinen Verständnis wurden die Anforderungen mit Hilfe von Schablonen formuliert. Dabei wurde festgelegt, welche Kriterien vorhanden sein müssen, sollten oder wünschenswert wären. Für eine spätere Evaluation wurden sie nach ihrer Wichtigkeit bewertet. Es stellten sich das Erfassen und Kartieren von Daten, die Offlinefunktionalität sowie die kostenfreie Nutzung der Tools als die wichtigsten Kriterien heraus. Anhand der zur Verfügung stehenden Literatur wurden drei Lösungsansätze für die Untersuchung festgelegt: Progressive Web Apps, Native Wrapper und lokaler Webserver. Der PWA-Ansatz wurde wegen der Aktualität und des Zukunftspotenzials ausgewählt. Native Wrapper stellen seit Jahren eine Möglichkeit der App Entwicklung dar, die auch eine Offlinefunktionalität anbieten und sie sich somit für einen Vergleich angeboten haben. Der Ansatz, auf einem mobilen Endgerät einen lokalen Webserver einzurichten, stellte sich als eine interessante Lösung für das Offlineproblem dar. Jedoch ist diese für die geforderte Verwendung aufgrund ihrer aufwändigen Installation nicht geeignet. Der Vergleich der Lösungsansätze wurde in Bezug auf die erarbeiteten Anforderungen ausgeführt. Um die Technologie zu bestimmen, die im praktischen Teil prototypisch umgesetzt werden sollte, wurden eine Nutzwert- und eine Prioritätenanalyse durchgeführt, wobei die Technologien abhängig von der Erfüllung der Anforderungen benotet wurden. Zusätzlich zur Benotung wurde die Gewichtung der Kriterien, die im Vorhinein entschieden wurde, angewendet. Die PWA ergab sich als die Technologie, die für den Anwendungsfall dieser Arbeit am besten geeignet ist. Die Umsetzung der Offline Web Mapping App wurde ausführlich dokumentiert, jedoch wurde der Fokus auf die wichtigsten Schritte der Programmierung gesetzt. Nach einer kurzen Beschreibung der Entwicklung der ursprünglichen Web App wurde Schritt für die Schritt die Migration zur Progressive Web App erläutert. Die Einrichtung hat sich aufgrund der geringen Vorkenntnisse in der JavaScript Programmierung als komplizierter herausgestellt als erwartet. Die Einrichtung der Offlinefunktionalität sowie die Erstellung der Datenbanken und die Speicherung der Hintergrundkarten haben mit Hilfe einer Einarbeitung ins Thema erfolgreich funktioniert. Die Einrichtung der GIS-Funktionen konnten aufgrund der Komplexität und der mangelnden Dokumentation nur teilweise eigenständig programmiert werden. Um eine Funktionstüchtigkeit der Applikation zu erreichen, wurden diese Funktionen mit Hilfe des Artikels "Using GeoJSON & indexedDB Together, GISCoding" [60] vervollständigt

7.2 Ausblick

In dieser Arbeit wurden drei Lösungsansätze, die einer Web Mapping Applikation eine Offlinefunktionalität verleihen können, anhand von Literatur und den erarbeiteten Anforderungen untersucht, verglichen und bewertet. Hierdurch wurden die Eigenschaften und Möglichkeiten einer Offline Applikation aufgezeigt. Der Fokus lag dabei auf einer Offline App, die zusätzlich GIS-Funktionen beherrscht und für die Datenerfassung geeignet ist. In der vorliegenden Thesis hat sich hierfür der PWA-Ansatz durchgesetzt. Für die anderen beiden Ansätze bedeutet das nur, dass sie für den Anwendungszweck dieser Arbeit weniger geeignet waren, viel mehr können sie sich in anderen Anwendungsfällen als besser herausstellen. Progressive Web Apps haben sich in den letzten Jahren stetig weiterentwickelt und es gilt abzuwarten, welche weiteren Entwicklungen in diesem Bereich noch entstehen. Für diesen Anwendungsfall haben sich PWAs als geeignet herausgestellt, jedoch sind sie noch nicht so weit, dass sie problemlos verwendet werden können. Die Einrichtung der Datenbank für eine Web Map ist noch nicht benutzerfreundlich und erfordert fortgeschrittene Kenntnisse in der JavaScript-Programmierung. Das Speichern der Hintergrundkarte auf dem Gerät ist für jemanden, der mit Geoinformationssystemen vertraut ist, nicht allzu kompliziert. Zukünftig werden die meisten Applikationen im Web verfügbar sein, da sie keine Installation benötigen und schnell abgerufen werden können. Mit dem "Offline-First-Prinzip" [34] werden seit einigen Jahren die Offlinefunktionalität und dessen Möglichkeiten untersucht. Im Moment gibt es noch nicht viele Lösungsansätze, jedoch wird sich das in den nächsten Jahren ändern. Dieses Thema ist aktuell und wird in Zukunft auch weitere Forschungsfragen hervorbringen. Eine Erweiterung für diese Arbeit würde darin bestehen, die Kartenkacheln direkt in der IndexedDB zu speichern. Dadurch könnte die Map direkt über die Datenbank abgerufen werden. Die Applikation würde vollständig offline im Browser laufen. Auch die Untersuchung von weiteren Speichermöglichkeiten wie die File System Access API [62] ist eine denkbare Erweiterung. Ein Ansatz wäre auch der Ausbau der Applikation um weitere Funktionen, wie zum Beispiel Tools aus dem technischen Zeichnen, um Gelände- und Oberflächenmodelle zu erstellen.

8 Literaturverzeichnis

[1] Deloitte Deutschland, *Smartphone Nutzung 2020: Studie zur Smartphone-Nutzung: Der deutsche Mobile Consumer im Profil.* [Online]. Verfügbar unter: https://www2.deloitte.com/de/de/pages/technology-media-and-telecommunications/articles/smartphone-nutzung-2020.html (Zugriff am: 14. Juli 2021).

[2] H. Heitkötter, S. Hanschke und T. A. Majchrzak, „Evaluating Cross-Platform Development Approaches for Mobile Applications“ in *Lecture Notes in Business Information Processing*, Bd. 140, *Web information systems and technologies: 8th international conference, WEBIST 2012, Porto, Portugal, April 18 - 21, 2012 ; revised selected papers*, J. Cordeiro und K.-H. Krempels, Hg., Berlin, Heidelberg: Springer, 2013, S. 120–138, doi: 10.1007/978-3-642-36608-6_8.

[3] T. Li, H. Du, L. Tang und Y. Xu, „The discussion of cross-platform mobile application based on Phonegap“ in *IEEE 4th International Conference on Software Engineering and Service Science (ICSESS)*, Beijing, China, 2013, S. 652–655, doi: 10.1109/ICSESS.2013.6615391.

[4] M. Ali und A. Mesbah, „Mining and characterizing hybrid apps“ in *FSE'16: 24nd ACM SIGSOFT International Symposium on the Foundations of Software Engineering*, Seattle WA USA, 2016, S. 50–56, doi: 10.1145/2993259.2993263.

[5] A. D. Brucker und M. Herzberg, „On the Static Analysis of Hybrid Mobile Apps“ in S. 72–88, doi: 10.1007/978-3-319-30806-7_5.

[6] J. Goetz und Y. Li, „Evaluation of Cross-Platform Frameworks for Mobile Applications“ in *International Journal of Engineering and Innovative Technology (IJEIT)*, 2018, S. 10–17. [Online]. Verfügbar unter: https://www.researchgate.net/publication/329488825_Evaluation_of_Cross-Platform_Frameworks_for_Mobile_Applications

[7] I. Dalmasso, S. K. Datta, C. Bonnet und N. Nikaein, „Survey, comparison and evaluation of cross platform mobile application development tools“ in *9th International Wireless Communications and Mobile Computing Conference (IWCMC 2013)*, Sardinia, Italy, 2013, S. 323–328, doi: 10.1109/IWCMC.2013.6583580.

[8] T. A. Majchrzak, B.-H. Andreas und G. Tor-Morten, „Progressive Web Apps: the Definite Approach to Cross-Platform Development?“ in *Hawaii International Conference on System Sciences*, 2018, S. 5735–5744, doi: 10.24251/HICSS.2018.718.

[9] I. Malavolta, K. Chinnappan, L. Jasmontas, S. Gupta und K. A. K. Soltany, „Evaluating the impact of caching on the energy consumption and performance of progressive web apps“ in *MOBILESoft '20: IEEE/ACM 7th International Conference on Mobile Software Engineering and Systems*, Seoul Republic of Korea, 2020, S. 109–119, doi: 10.1145/3387905.3388593.

[10] I. Malavolta, G. Procaccianti, P. Noorland und P. Vukmirovic, „Assessing the Impact of Service Workers on the Energy Efficiency of Progressive Web Apps“ in *IEEE/ACM 4th International Conference on Mobile Software Engineering and Systems (MOBILESoft)*, Buenos Aires, Argentina, 2017, S. 35–45, doi: 10.1109/MOBILESoft.2017.7.

[11] J. Lee, H. Kim, J. Park, I. Shin und S. Son, „Pride and Prejudice in Progressive Web Apps“ in *CCS '18: 2018 ACM SIGSAC Conference on Computer and Communications Security*, Toronto Canada, 2018, S. 1731–1746, doi: 10.1145/3243734.3243867.

[12] Google Developers, *Progressive Web Apps.* [Online]. Verfügbar unter: https://web.dev/progressive-web-apps/ (Zugriff am: 14. Juli 2021).

[13] P. LePage und S. Richard, *What are Progressive Web Apps?* [Online]. Verfügbar unter: https://web.dev/what-are-pwas/ (Zugriff am: 15. Juli 2021).

[14] C. Liebel, „Progressive Web Apps, Teil 5: Das App-Modell der Zukunft?“, *heise online*, 10. Juli 2017, 2017. [Online]. Verfügbar unter: https://www.heise.de/developer/artikel/Progressive-Web-Apps-Teil-5-Das-App-Modell-der-Zukunft-3767383.html. Zugriff am: 14. Juli 2021.

[15] S. Springer, *Offlinemodus: Die Achillesferse der Progressive Web Apps?* [Online]. Verfügbar unter: https://entwickler.de/programmierung/offlinemodus-die-achillesferse-der-progressive-web-apps/ (Zugriff am: 14. Juli 2021).

[16] P. LePage und S. Richard, *What makes a good Progressive Web App?* [Online]. Verfügbar unter: https://web.dev/pwa-checklist/ (Zugriff am: 8. März 2022).

[17] T. Ater, *Building Progressive Web Apps: Bringing the Power of Native to the Browser,* 1. Aufl. Beijing: O´Reilly, 2017. [Online]. Verfügbar unter: https://ebookcentral.proquest.com/lib/kxp/detail.action?docID=5045024

[18] J. M. Wargo, *Learning progressive web apps: Building modern web apps using Service Workers*. Boston: Addison-Wesley/Pearson Education, 2020.

[19] C. Love, *Progressive Web Application Development by Example: Develop Fast, Reliable, and Engaging User Experiences for the Web*. Birmingham: Packt Publishing Ltd, 2018. [Online]. Verfügbar unter: https://ebookcentral.proquest.com/lib/gbv/detail.action?docID=5477666

[20] A. Neumann, „Web Mapping and Web Cartography“ in *Springer Handbook of Geographic Information*, W. Kresse und D. M. Danko, Hg., Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, S. 567–586, doi: 10.1007/978-3-540-72680-7_14.

[21] M. Dorman, *Introduction to web mapping*. Boca Raton, FL: CRC Press, 2020. [Online]. Verfügbar unter: https://www.taylorfrancis.com/books/9780429352874

[22] StatCounter Global Stats, *Mobile Operating System Market Share Worldwide.* [Online]. Verfügbar unter: https://gs.statcounter.com/os-market-share/mobile/worldwide (Zugriff am: 1. Februar 2022).

[23] J. Nielsen und R. Budiu, *Mobile usability: Für iPhone, iPad, Android und Kindle,* 1. Aufl. Heidelberg, München, Landsberg, Frechen, Hamburg: mitp, 2013. [Online]. Verfügbar unter: https://books.google.at/books?id=vDxST9fHBJcC

[24] IBM, *Native, web or hybrid mobile-app development.* [Online]. Verfügbar unter: https://docplayer.net/16266542-Native-web-or-hybrid-mobile-app-development.html (Zugriff am: 20. Januar 2022).

[25] W. Jobe, „Native Apps Vs. Mobile Web Apps“, *Int. J. Interact. Mob. Technol.*, Jg. 7, Nr. 4, S. 27–32, 2013, doi: 10.3991/ijim.v7i4.3226.

[26] D. Sin, E. Lawson und K. Kannoorpatti, „Mobile Web Apps - The Non-programmer's Alternative to Native Applications“ in *5th International Conference on Human System Interactions (HSI)*, Perth, Australia, 2012, S. 8–15, doi: 10.1109/HSI.2012.11.

[27] Evoke Technologies, *How to Build Offline-first Mobile Apps Using Cordova Hybrid Platform.* [Online]. Verfügbar unter: https://www.evoketechnologies.com/blog/offline-first-mobile-apps-cordova-hybrid-platform/ (Zugriff am: 14. Dezember 2021).

[28] S. Augsten, „Was ist eine Cross-Platform App?“, *Dev-Insider*, 7. Feb. 2020, 2020. [Online]. Verfügbar unter: https://www.dev-insider.de/was-ist-eine-cross-platform-app-a-898699/. Zugriff am: 8. Juni 2022.

[29] D. Berger-Grabner, *Wissenschaftliches Arbeiten in den Wirtschafts- und Sozialwissenschaften: Hilfreiche Tipps und praktische Beispiele,* 3. Aufl. Wiesbaden: Springer Gabler, 2016. [Online]. Verfügbar unter: http://www.springer.com/

[30] S. Vogl, „Gruppendiskussion“ in *Handbuch Methoden der empirischen Sozialforschung*, N. Baur und J. Blasius, Hg., Wiesbaden: Springer Fachmedien Wiesbaden, 2019, S. 695–700, doi: 10.1007/978-3-658-21308-4_46.

[31] C. Rupp, *Requirements-Engineering und -Management: Das Handbuch für Anforderungen in jeder Situation,* 7. Aufl. München: Hanser, 2021.

[32] BMI, *Organisationshandbuch - Qualitative Bewertungsmethoden.* [Online]. Verfügbar unter: https://www.orghandbuch.de/OHB/DE/Organisationshandbuch/6_MethodenTechniken/65_Wirtschaftlichkeitsuntersuchung/652_Qualitative/qualitative-node.html (Zugriff am: 9. Juni 2022).

[33] D. Sauble, *Offline first web development: Design and implement a robust offline app experience using Sench Touch and PouchDB*. Birmingham, UK: Packt Publishing, 2015. [Online]. Verfügbar unter: http://proquest.tech.safaribooksonline.de/9781785884573

[34] A. Feyerke, *Designing Offline-First Web Apps.* [Online]. Verfügbar unter: https://alistapart.com/article/offline-first/ (Zugriff am: 14. Juli 2021).

[35] A. Russell, „Progressive Web Apps: Escaping Tabs Without Losing Our Soul“, *Alex Russell*, 15. Juni 2015, 2015. Zugriff am: 3. März 2022.

[36] S. Cox, „Progressive Web Apps: A Conversation with Google’s Alex Russell“, *Lumavate*, 28. Jan. 2019, 2019. [Online]. Verfügbar unter: https://medium.com/lumavate/progressive-web-apps-a-conversation-with-googles-alex-russell-ccaf15f33260. Zugriff am: 3. März 2022.

[37] *Introduction to progressive web apps - Progressive web apps (PWAs) | MDN.* [Online]. Verfügbar unter: https://developer.mozilla.org/en-US/docs/Web/Progressive_web_apps/Introduction#advantages_of_web_applications (Zugriff am: 9. Juni 2022).

[38] *Service-Worker benutzen - Web API Referenz | MDN.* [Online]. Verfügbar unter: https://developer.mozilla.org/de/docs/Web/API/Service_Worker_API/Using_Service_Workers (Zugriff am: 6. Dezember 2021).

[39] S. Springer, *Offline First ist das neue Mobile First.* [Online]. Verfügbar unter: https://entwickler.de/webentwicklung/offline-first-ist-das-neue-mobile-first (Zugriff am: 10. September 2021).

[40] M. Gaunt, *Service Workers: an Introduction.* [Online]. Verfügbar unter: https://developers.google.com/web/fundamentals/primers/service-workers/ (Zugriff am: 14. Juli 2021).

[41] Web App Manifest, *Web App Manifest | MDN.* [Online]. Verfügbar unter: https://developer.mozilla.org/de/docs/Web/Manifest (Zugriff am: 31. Dezember 2021).

[42] *Web Storage API - Web API Referenz | MDN.* [Online]. Verfügbar unter: https://developer.mozilla.org/de/docs/Web/API/Web_Storage_API (Zugriff am: 8. Dezember 2021).

[43] *IndexedDB - Web API Referenz | MDN.* [Online]. Verfügbar unter: https://developer.mozilla.org/de/docs/Web/API/IndexedDB_API (Zugriff am: 4. Januar 2022).

[44] Apache Cordova, *Apache Cordova.* [Online]. Verfügbar unter: https://cordova.apache.org/ (Zugriff am: 9. Juni 2022).

[45] Capacitor, *Capacitor by Ionic - Cross-platform apps with web technology.* [Online]. Verfügbar unter: https://capacitorjs.com/ (Zugriff am: 2. Januar 2022).
[46] Electron, *Electron | Build cross-platform desktop apps with JavaScript, HTML, and CSS.* [Online]. Verfügbar unter: https://www.electronjs.org/ (Zugriff am: 5. Januar 2022).
[47] Termux, *Termux Wiki.* [Online]. Verfügbar unter: https://wiki.termux.com/wiki/Main_Page (Zugriff am: 15. Dezember 2021).
[48] M. Lang, „Termux samt API-Zugriff einrichten“, *Dev-Insider*, 9. Dez. 2019, 2019. [Online]. Verfügbar unter: https://www.dev-insider.de/termux-samt-api-zugriff-einrichten-a-887006/. Zugriff am: 4. Januar 2022.
[49] *Background Synchronization API - Web APIs | MDN.* [Online]. Verfügbar unter: https://developer.mozilla.org/en-US/docs/Web/API/Background_Synchronization_API (Zugriff am: 10. Juni 2022).
[50] Google Developers, *Offline Data.* [Online]. Verfügbar unter: https://web.dev/learn/pwa/offline-data/ (Zugriff am: 3. Januar 2022).
[51] L. Vu, „How to Access the Camera in a PWA“, *SimiCart*, 19. Aug. 2020, 2020. [Online]. Verfügbar unter: https://www.simicart.com/blog/pwa-camera-access/. Zugriff am: 2. Januar 2022.
[52] M. Funk, „Hybrid-Apps und ihre Performance – ein viel diskutiertes Thema“, *FLYACTS*, 22. Jan. 2015, 2015. [Online]. Verfügbar unter: https://www.flyacts.com/hybrid-apps-und-ihre-performance-ein-viel-diskutiertes-thema. Zugriff am: 3. Januar 2022.
[53] N. Reimers, *Wie viel Leistung braucht ein Webserver? – Webhosting Vergleich.* [Online]. Verfügbar unter: https://www.webhosterwissen.de/know-how/server/wie-viel-leistung-braucht-ein-webserver/ (Zugriff am: 3. Januar 2022).
[54] A. Osmani, „Offline Storage for Progressive Web Apps - Dev Channel - Medium“, *Dev Channel*, 15. Aug. 2016, 2016. [Online]. Verfügbar unter: https://medium.com/dev-channel/offline-storage-for-progressive-web-apps-70d52695513c#.lm82vlyt8. Zugriff am: 3. Januar 2022.
[55] *JavaScript | MDN.* [Online]. Verfügbar unter: https://developer.mozilla.org/en-US/docs/Web/JavaScript (Zugriff am: 13. Juni 2022).
[56] *CSS | MDN.* [Online]. Verfügbar unter: https://developer.mozilla.org/en-US/docs/Web/CSS (Zugriff am: 13. Juni 2022).
[57] *HTML: HyperText Markup Language | MDN.* [Online]. Verfügbar unter: https://developer.mozilla.org/en-US/docs/Web/HTML (Zugriff am: 13. Juni 2022).
[58] *OpenLayers - Welcome.* [Online]. Verfügbar unter: https://openlayers.org/ (Zugriff am: 13. Juni 2022).
[59] *Using Service Workers - Web APIs | MDN.* [Online]. Verfügbar unter: https://developer.mozilla.org/en-US/docs/Web/API/Service_Worker_API/Using_Service_Workers (Zugriff am: 12. Juli 2022).
[60] GISLayer Articles, „Using GeoJSON & indexedDB Together, GISCoding - GISLayer Articles - Medium“, *Medium*, 16. Jan. 2021, 2021. [Online]. Verfügbar unter: https://gislayer.medium.com/using-geojson-indexeddb-together-giscoding-7f1d53df9abe. Zugriff am: 3. Januar 2022.
[61] *How do you set up a local testing server? - Learn web development | MDN.* [Online]. Verfügbar unter: https://developer.mozilla.org/en-US/docs/Learn/Common_questions/set_up_a_local_testing_server (Zugriff am: 13. Juni 2022).

[62] P. LePage und T. Steiner, *The File System Access API: simplifying access to local files.* [Online]. Verfügbar unter: https://web.dev/file-system-access/ (Zugriff am: 3. Januar 2022).

Anhang I: Skripte zur Realisierung der Beispielapplikation

index.html

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>Openlayersmap</title>
    <link rel="stylesheet" href="lib/openlayers/ol.css" type="text/css">
    <script src="lib/openlayers/ol.js"></script>

    <style>
    .map {
        width: 100%;
        height: 400px;
    }
    </style>

</head>

<body>
    Openlayersmap
    <div id="map" class="map"></div>
    <script src="init.js"></script>
</body>
</html>
```

Init.js

```
console.log("Openlayersmap");
// add a XYZ layer to our map
var positron = new ol.layer.Tile({
        title: 'Carto Positron Basemap',
        source: new ol.source.XYZ({
            url: 'https://basemaps.cartocdn.com/light_all/{z}/{x}/{y}.png',
            attributions: "Maps: © <a href='https://carto.com/attribution/'
target='_blank'>CARTO</a> | <a href='https://openstreetmap.org/copyright'
target='_blank'>© OpenStreetMap</a>-contributors | "
        })
    });

// This is my map
var map = new ol.Map({
```

```
    controls:
        ol.control.defaults(),
    layers: [
        positron
    ],
    target: 'map',
    view: new ol.View({
        center: [1822578.81, 6141587.78],
        zoom: 12
    }
    )
});
```

Anhang II: Skripte zur Realisierung der Progressive Web App

Index.html

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Progressive Web Mapping App</title>
    <link rel="stylesheet" href="lib/openlayers/ol.css" type="text/css">
    <link rel="manifest" href="/manifest.json">
    <link rel="icon" href="/favicon.ico" type="image/x-icon">
    <script src="lib/openlayers/ol.js"></script>

    <style>
    .map {
        width: 100%;
        height: 1500px;
    }

    .topRight{
      position: absolute;
      top: 90px;
      right: 0px;
      z-index: 2;
    }

    button{
      display: block;
      width: 60%;
      height: 40px;
      color: rgb(255, 255, 255);
      background-color: rgb(40, 118, 244);
      border: 1px solid rgb(1, 19, 1);
      border-color:#0e0101;
      cursor: pointer;
      transition-duration: 0,4s;
    }

    button:hover {
      background-color: rgb(102, 163, 242);
      color:rgba(0, 0, 0, 0.903);
    }
```

```
    .locate {
      top: 4em;
      left: .5em;
    }

    h1 {
      text-align:center;

    }

    </style>

</head>

<body>
    <h1><u>Overview of the city centre of Vienna</u></h1>
    <div id="map" class="map"></div>
    <div class="topRight">
      <button onclick="draw('Point')" title="Draw Point"><font
size="5"><b>&#8408</b></font></button>
      <button onclick="draw('LineString')" title="Draw Line"><font
size="5"><b>&#8212</b></font></button>
      <button onclick="draw('Polygon')" title="Draw Polygon"><font
size="5"><b>&#916</b></font></button>
      <button onclick="readGeoJSON()" title="Add a GeoJSON file"><b>Add
GeoJSON</b></button>
      <button onclick="removeGeoJSON()" title="Remove GeoJSON file"
id="removeButton"><b>Remove GeoJSON</b></button>
      <button onclick="clearAll()" title="Clear all data"><b>Clear
All</b></button>
    </div>

    <script src="app.js"></script>
    <script src="map.js"></script>
    <script src="indexedDB.js"></script>

</body>
</html>
```

app.js (in Anlehnung an: [59])

```
if ('serviceWorker' in navigator) {
    navigator.serviceWorker.register('/sw.js').then(function(reg) {
      // Registration successful
      console.log('Registration successful. Scope ist ' + reg.scope);
    }).catch(function(error) {
      // Registration failed
```

```
      console.log('Registration failed with ' + error);
    });
  };
```

sw.js (in Anlehnung an: [59])

```
self.addEventListener("install", function(event) {
  event.waitUntil(
    caches.open("geodata").then(function(cache) {
      return cache.add("/index_offline.html");
    })
  );
});

self.addEventListener("fetch", function(event) {
  event.respondWith(
    fetch(event.request).catch(function() {
      return caches.match("/index_offline.html");
    })
  );
});
```

map.js (in Anlehnung an:[60])

```
//Basemap
var map = new ol.Map({
    target: 'map',
    layers: [
      new ol.layer.Tile({
        source: new ol.source.XYZ({
          url: 'http://localhost:8000/tiles/{z}/{x}/{y}.png', //Enter the
folder path of the rastertiles
          crossOrigin: 'anonymous'
        })
      })
    ],
    //Map section and zoom level
    view: new ol.View({
      center: ol.proj.fromLonLat([16.37326496188555,48.20842614255267]),
//Enter the coordinates for the map section
      zoom: 18
    })
  });

  const source = new ol.source.Vector();
const layer = new ol.layer.Vector({
  source: source
```

```
});
map.addLayer(layer);


//add geolocation
navigator.geolocation.watchPosition(function(pos) {
  const coords = [pos.coords.longitude, pos.coords.latitude];
  const accuracy = ol.geom.Polygon.circular(coords, pos.coords.accuracy);
  source.clear(true);
  source.addFeatures([
    new ol.Feature(accuracy.transform('EPSG:4326',
map.getView().getProjection())),
    new ol.Feature(new ol.geom.Point(ol.proj.fromLonLat(coords)))
  ]);
}, function(error) {
  alert(`ERROR: ${error.message}`);
}, {
  enableHighAccuracy: true
});

const locate = document.createElement('div');
locate.className = 'ol-control ol-unselectable locate';
locate.innerHTML = '<button title="Locate me">◎</button>';
locate.addEventListener('click', function() {
  if (!source.isEmpty()) {
    map.getView().fit(source.getExtent(), {
      maxZoom: 18,
      duration: 500
    });
  }
});
map.addControl(new ol.control.Control({
  element: locate
}));

  //Creation of Layer 1 for the GeoJSON files
  var layer1 = new ol.layer.Vector({
    source: new ol.source.Vector(),
    style: new ol.style.Style({
      fill: new ol.style.Fill({
        color: 'rgba(139, 195, 74, 0.5)'
      }),
      stroke: new ol.style.Stroke({
        color: '#8BC34A',
        width: 3
      }),
      image: new ol.style.Circle({
```

```
      radius: 5,
      fill: new ol.style.Fill({
        color: '#8BC34A'
      })
    })
  })
});
map.addLayer(layer1);

//Creation of layer 2 for the vectordata
var layer2 = new ol.layer.Vector({
  source: new ol.source.Vector(),
  style: new ol.style.Style({
    fill: new ol.style.Fill({
      color: 'rgba(255, 193, 7, 0.7)'
    }),
    stroke: new ol.style.Stroke({
      color: '#00bcd4',
      width: 4
    }),
    image: new ol.style.Circle({
      radius: 7,
      fill: new ol.style.Fill({
        color: '#9c27b0'
      })
    })
  })
});
map.addLayer(layer2);

function clearLayer(layer) {
  layer.getSource().clear();
}
//Function for the adding of the GeoJSON files
function featuresToGeoJSON(features){
  var reader = new ol.format.GeoJSON();
  var geojson = reader.writeFeatures(features,{
    featureProjection: 'EPSG:3857',
    dataProjection: 'EPSG:4326'
  });
  return JSON.parse(geojson);
}

function GeoJSONToFeature(geojson){
  var reader = new ol.format.GeoJSON();
  var features = reader.readFeatures(geojson,{
    featureProjection: 'EPSG:3857',
```

```
    dataProjection: 'EPSG:4326'
  });
  return features;
}

function addFeaturesToLayer(layer,features) {
  var src = layer.getSource();
  src.clear();
  src.addFeatures(features);
}

function zoomToLayer(layer) {
  var features = layer.getSource().getFeatures();
  if(features.length>0){
    var extend = layer.getSource().getExtent();
    map.getView().fit(extend, map.getSize());
    var zoomlevel = map.getView().getZoom();
    if (zoomlevel > 18) {
      zoomlevel = 18;
    }
    map.getView().setZoom(zoomlevel);
  }
}
//Function for drawing polygons, lines und points
function draw(type){
  var draw = new ol.interaction.Draw({
    source: layer2.getSource(),
    type: type
  });
  map.addInteraction(draw);
  draw.on('drawend', function (e) {
    setTimeout(function(){
      var features = layer2.getSource().getFeatures();
      var geojson = featuresToGeoJSON(features);
      database_1.add({
        type:'updateLayer',
        param:{
          layerId:'layer2',
          geojson:geojson
        }});

    },100)
    map.removeInteraction(draw);
  });
}
//Function for reading the GeoJSON files
function readGeoJSON(){
```

```
    var input = document.createElement('input');
    input.type='file';
    input.accept='.json, .geojson';
    input.addEventListener('input', function (e) {
      var file = e.target.files[0];
      var reader = new FileReader();
      reader.addEventListener("load", function (e2) {
        var geojson = JSON.parse(reader.result);
        database_1.add({
          type:'addLayer',
          param:{
            layerId:'layer1',
            geojson:geojson
          },
          callback:function(status1){
            if(status1){
              var features = GeoJSONToFeature(geojson);
              addFeaturesToLayer(layer1,features);
              zoomToLayer(layer1);
              document.getElementById('removeButton').style.display='inline
-block';
            }else{
              database_1.add({
                type:'deleteLayer',
                param:{
                  layerId:'layer1'
                },
                callback:function(status2){
                  if(status2){
                    document.getElementById('removeButton').style.display='
none';
                    database_1.add({
                      type:'addLayer',
                      param:{
                        layerId:'layer1',
                        geojson:geojson
                      },
                      callback:function(status3){
                        if(status3){
                          var features = GeoJSONToFeature(geojson);
                          addFeaturesToLayer(layer1,features);
                          zoomToLayer(layer1);
                          document.getElementById('removeButton').style.dis
play='inline-block';
                        }
                      }
                    });
```

```
                }
              }});
          }
        }
      });
    });
    reader.readAsText(file);
  });
  input.click();
}
//Function for the removal of GeoJSON files
function  removeGeoJSON() {
  database_1.add({
    type:'deleteLayer',
    param:{
      layerId:'layer1'
    },
    callback:function(status){
      if(status){
        document.getElementById('removeButton').style.display='none';
        clearLayer(layer1);
      }
    }});
}

function clearAll() {
  database_1.add({
    type:'updateLayer',
    param:{
      layerId:'layer1',
      geojson:{ "type": "FeatureCollection", "features": [] }
    },
    callback:function(status){
      if(status){
        clearLayer(layer1);
      }
  }})
  database_1.add({
    type:'updateLayer',
    param:{
      layerId:'layer2',
      geojson:{ "type": "FeatureCollection", "features": [] }
    },
    callback:function(status){
      if(status){
        clearLayer(layer2);
      }
```

```
    }})
  }
```

Indexeddb.js (in Anlehnung an: [60])

```
//Creation of the database and saving files in the IndexedDB

window.indexedDB = window.indexedDB || window.mozIndexedDB ||
window.webkitIndexedDB || window.msIndexedDB;
window.IDBTransaction = window.IDBTransaction ||
window.webkitIDBTransaction || window.msIDBTransaction;
window.IDBKeyRange = window.IDBKeyRange || window.webkitIDBKeyRange ||
window.msIDBKeyRange;

class IDBGeoJSON {
  constructor(params){
    this.db_name = params.db_name;
    this.localStorageName = this.db_name+'-indexedDB-version';
    this.round=parseInt(params.round) || 5;
    this.jobs=[];
    this.add({
      type:'createDB',
      prop:{
        db_name:this.db_name
      }
    })
  }

  setVersion(version){
    localStorage.setItem(this.localStorageName, version);
  }

  getVersion(){
    var version = localStorage.getItem(this.localStorageName);
    if (version == null || version == '') {
      localStorage.setItem(this.localStorageName, 1);
    }
    version = localStorage.getItem(this.localStorageName);
    return parseInt(version,10);
  }

  createDB(callback){
    var self = this;
    var request = indexedDB.open(this.db_name);
    request.onerror = function(event) {
      console.log('indexedDB Error Code:'+event.target.errorCode);
```

```
      callback(false)
    };
    request.onsuccess = function(sender) {
      var db = sender.target.result;
      var newVersion = db.version+1;
      self.setVersion(newVersion);
      var tableNames = db.objectStoreNames;
      db.close();
      if(tableNames.length==0){
        var request = indexedDB.open('geodata', newVersion);
        request.onupgradeneeded = function (event) {
          var db2 = event.target.result;
          var layerTable = db2.createObjectStore('layers', {
            keyPath: "layerid"
          });
          layerTable.createIndex('layerid', 'layerid', {
            unique: true
          });
          layerTable.createIndex('geojson', 'geojson', {
            unique: false
          });
          db2.close();
          callback(true);
        }
      }else{
        callback(true);
      }
    };
  }

  deleteLayer(layerId,callback){
    var self = this;
    var version = this.getVersion();
    var request = window.indexedDB.open('geodata', version);
    request.onsuccess = function (event) {
      var db = event.target.result;
      var request2 = db.transaction(['layers'],
"readwrite").objectStore('layers').delete(layerId);
      request2.onsuccess = function (event) {
        var db = event.target.transaction.db;
        var currentVersion = db.version;
        db.close();
        self.setVersion(currentVersion);
        callback(true);
      }
      request.onerror = function(event) {
        console.log('indexedDB Error Code:'+event.target.errorCode);
```

```
        callback(false)
      };
    }
  }

  addLayer(layerId,GeoJSON,callback){
    var self = this;
    var version = self.getVersion();
    var request = indexedDB.open(this.db_name, version);
    request.onsuccess = function(event){
      var db = event.target.result;
      var data = {
        layerid:layerId,
        geojson:GeoJSON
      };
      var insert = db.transaction(['layers'],
"readwrite").objectStore('layers').add(data);
      insert.onsuccess = function (e1) {
        event.target.result.close();
        callback(true);
      }
      insert.onerror = function (e1) {
        event.target.result.close();
        console.log(e1.target.error.message);
        callback(false);
      }
      db.close();
    }
  }

  updateLayer(layerId,GeoJSON,callback){
    var version = this.getVersion();
    var newVersion = version + 1;
    this.setVersion(newVersion);
    var request = window.indexedDB.open('geodata', newVersion);
    request.onsuccess = function (event) {
      var db = event.target.result;
      var store = db.transaction(['layers'],
"readwrite").objectStore('layers');
      var data = {
        layerid: layerId,
        geojson: GeoJSON,
      };
      store.put(data);
      db.close();
      callback(true);
    }
```

```
    request.onerror = function (e1) {
      var db = e1.target.result;
      db.close();
      console.log(e1.target.error.message);
      callback(false);
    }
  }

  showLayer(callback){
    var version = this.getVersion();
    var newVersion = version + 1;
    this.setVersion(newVersion);
    var request = window.indexedDB.open('geodata', newVersion);
    request.onsuccess = function (event) {
      var db = event.target.result;
      var store = db.transaction(['layers'],
"readonly").objectStore('layers');
      var result = [];
      store.openCursor().onsuccess = function (e) {
        var cursor = e.target.result;
        if (cursor) {
          result.push(cursor.value);
          cursor.continue();
        }else{
          callback(result);
        }
      }
      db.close();
    }
    request.onerror = function (e1) {
      var db = e1.target.result;
      db.close();
      console.log(e1.target.error.message);
      callback(false);
    }
  }

  add(job){
    this.jobs.push(job);
    if(this.jobs.length==1){
      this.next();
    }
  }

  next(){
    var self = this;
    if(this.jobs.length>0){
```

```
      const job = this.jobs[0];
      switch(job.type){
        case 'createDB':{
          this.createDB((status)=>{
            if(job.callback!==undefined){
              job.callback(status);
            }
            self.jobs.splice(0,1);
            self.next();
          })
          break;
        }
        case 'addLayer':{
          this.addLayer(job.param.layerId,job.param.geojson,(status)=>{
            if(job.callback!==undefined){
              job.callback(status);
            }
            self.jobs.splice(0,1);
            self.next();
          });
          break;
        }
        case 'deleteLayer':{
          self.deleteLayer(job.param.layerId,function(status){
            if(job.callback!==undefined){
              job.callback(status);
            }
            self.jobs.splice(0,1);
            self.next();
          });
          break;
        }
        case 'updateLayer':{
          self.updateLayer(job.param.layerId,job.param.geojson,function(sta
tus){
            if(job.callback!==undefined){
              job.callback(status);
            }
            self.jobs.splice(0,1);
            self.next();
          });
          break;
        }
        case 'showLayer':{
          self.showLayer(function(data){
            if(job.callback!==undefined){
              job.callback(data);
```

```
            }
            self.jobs.splice(0,1);
            self.next();
          });
          break;
        }
      }
    }
  }

}

var database_1 = new IDBGeoJSON({
  db_name:'geodata'
});

database_1.add({
  type:'addLayer',
  param:{
    layerId:'layer2',
    geojson:{ "type": "FeatureCollection", "features": [] }
  }
});

database_1.add({
  type:'showLayer',
  callback:(layers)=>{
    layers.map((layer)=>{
      var features = GeoJSONToFeature(layer.geojson);
      addFeaturesToLayer(window[layer.layerid],features);
      zoomToLayer(window[layer.layerid]);
    });
  }
});
```

Anhang III: Prioritätenanalyse

Priorisierung der Kriterien

Kriterien	ANF4	ANF8	ANF3	ANF13	ANF9	ANF11	ANF10	ANF7	ANF5	ANF12	ANF1	ANF2	ANF6	ANF14	Summe/Kriterium	Gewichtung (%)
ANF4		3	2	3	2	3	3	3	3	3	1	1	3	3	33	9,1%
ANF8	1		1	3	3	3	3	1	1	3	1	1	1	3	25	6,9%
ANF3	2	3		3	3	3	3	3	3	3	1	1	3	3	34	9,3%
ANF13	1	1	1		1	1	1	1	1	2	1	1	1	2	15	4,1%
ANF9	2	1	1	3		2	2	1	2	3	1	1	2	3	24	6,6%
ANF11	1	1	1	3	2		2	1	1	2	1	1	1	3	20	5,5%
ANF10	1	1	1	3	2	2		2	1	2	1	1	1	3	21	5,8%
ANF7	1	3	1	3	3	3	2		1	2	1	1	2	3	26	7,1%
ANF5	1	3	1	3	2	3	3	3		3	1	1	2	3	29	8,0%
ANF12	1	1	1	2	1	2	2	2	1		1	1	1	3	19	5,2%
ANF1	3	3	3	3	3	3	3	3	3	3		2	3	3	38	10,4%
ANF2	3	3	3	3	3	3	3	3	3	3	2		3	3	38	10,4%
ANF6	1	3	1	3	2	3	3	2	2	3	1	1		3	28	7,7%
ANF14	1	1	1	2	1	1	1	1	1	1	1	1	1		14	3,8%
Gesamt															364	100,0%

Tabelle 16: Priorisierung der Kriterien (Eigene Darstellung)

Erfassungstätigkeiten

Kriterien	ANF4-A	ANF4-B	Summe/Kriterium	Gewichtung
ANF4-A		2	2	**4,53%**
ANF4-B	2		2	**4,53%**
Gesamt			**4**	**9,1%**

Tabelle 17: Priorisierung von ANF 4 (Eigene Darstellung)

Performance

Kriterien	ANF7-A	ANF7-B	Summe/Kriterium	Gewichtung
ANF7-A		3	3	**5,36%**
ANF7-B	1		1	**1,79%**
Gesamt			**4**	**7,1%**

Tabelle 18: Priorisierung von ANF 7 (Eigene Darstellung)

Aufwand

Kriterien	ANF8-A	ANF8-B	ANF8-C	ANF8-D	ANF8-E	Summe/ Kriterium	Gewichtung
ANF8-A		3	3	1	2	9	**1,38%**
ANF8-B	1		2	3	2	8	**1,22%**
ANF8-C	1	2		3	2	8	**1,22%**
ANF8-D	3	3	1		3	10	**1,53%**
ANF8-E	2	2	2	1		7	**1,07%**
Gesamt						**42**	**6,43%**

Tabelle 19: Priorisierung von ANF 8 (Eigene Darstellung)

Fotos

Kriterien	ANF11-A	ANF11 -B	Summe/Kriterium	Gewichtung
ANF11-A	0	3	3	**4,12%**
ANF11-B	1	0	1	**1,37%**
Gesamt			**4**	**5,5%**

Tabelle 20: Priorisierung von ANF 12 (Eigene Darstellung)

Anhang IV: Übersicht über die Technologien

Anforderungen	PWA	Cordova	Webserver
Usecase			
Erfassung und Kartierung	Einrichtung möglich	Wenn in der Web App enthalten	Wenn in der Web App enthalten
Lizenz	W3C	Apache 2.0	GPLv3
Offlinefunktionalität	ja	ja	ja
Intranet	ja	ja	ja
Datenmanagement			
Speicherung und Lesen von Daten	Cache und Indexed DB	Lokaler Speicher	Lokaler Speicher
Synchronisation	Über Online-Zugang	Über Online-Zugang	Über Online-Zugang
Formate	Vektordaten, Rasterdaten, Bilder, Videos	Vektordaten, Rasterdaten, Bilder, Videos	Vektordaten, Rasterdaten, Bilder, Videos
Fotos (Speicherung, Verlinkung)	möglich	Mit Kamera Plugin	Mit Kamera Plugin
Setup			
Aufwand Einrichtung	- Entwicklung einer Web App - Installation von Service Worker und Web App Manifest - Einrichtung der IndexedDB	- Installation zusätzlicher Software notwendig - Abhängig von der Plattform (Android, iOS, Windows)	Download der Termux App und Installation auf dem Gerät
Update	Neustart der Web App	Plugin für ein automatisches Update	Über einen Eingabebefehl
Funktionalität			
Performance	Gute Performance durch den Abruf von Daten direkt aus der Datenbank	Die Performance hängt vom Browser und dem Gerät ab	Nicht prüfbar

Kompatibilität	Desktop: Edge, Firefox, Safari, Chrome, Opera Mobil: Android, iOS, Windows[38]	Mobil: Android, iOS	Mobil: Android
Geräte	Desktop, mobile Geräte	Desktop, mobile Geräte (Empfohlen)	mobile Geräte
Kamera	Zugriff möglich	Zugriff über Plugin	Zugriff über Plugin

Tabelle 21: Übersicht über die Lösungsansätze (Eigene Darstellung)

[38] https://caniuse.com/serviceworkers (Zugriff: 18.12.21)