Ostbayerische Technische Hochschule Amberg-Weiden
Fakultät Elektrotechnik, Medien und Informatik

Studiengang Medieninformatik

Studienarbeit App-Programmierung WiSe 2022/23

von

Annika **Stadelmann**

Entwicklung eines Parkleitsystems

Bearbeitungszeitraum:	von	22. November 2022
	bis	17. Januar 2023

Inhaltsverzeichnis

Kapitel 1

Einleitung

Im Rahmen der Studienarbeit sollte eine App als Parkleitsystem für die neun Parkhäuser um den Altstadtring in Amberg entwickelt werden. Neben der Anzeige der Parkflächen auf einer Karte, war es wichtig, auch detaillierte Informationen anzuzeigen. Dabei handelte es sich um Kosten pro Stunde, den Namen des Parkhauses oder Parkplatzes, die Anzahl der verfügbaren, belegten oder gesamten Parkplätze, sowie ein aufsteigender, gleichbleibender oder fallender Trend. Des Weiteren muss dem Nutzer auch mitgeteilt werden, ob und wann das Parkhaus geöffnet ist und ob es überhaupt befahren werden kann.

Neben der Anzeige der Daten sollte es auch die Möglichkeit geben, eine Navigation zur Parkmöglichkeit zu starten, sobald der Nutzer sich in einem bestimmten Radius um die Örtlichkeit befindet. Wenn das sogenannte Geofence betreten wird, soll neben der Navigationsmöglichkeit als Toast auch noch Text als Sprachausgabe ausgegeben werden, um den Nutzer darauf hinzuweisen, welches Parkhaus am nächsten ist und befahren werden könnte. Damit der Nutzer nun nicht durch unpassende Sprachausgabe in unangenehme Situationen gebracht wird, sollte die Möglichkeit gegeben sein, den Ton auszuschalten. Außerdem sollen die Parkmöglichkeiten auch als Liste angezeigt und favorisiert werden können.

Um die aktuellen Parkhausdaten zu bekommen, soll eine API verwendet werden, welche die Daten bereitstellt und regelmäßig aktualisiert. Die Daten sind im XML-Format und hier zu finden: `https://parken.amberg.de/wp-content/uploads/pls/pls.xml`

Im Folgenden Verlauf dieses Dokuments werden zunächst die verwendeten Technologien beschrieben. Anschließend werden die Vorüberlegungen zur Architektur dargelegt und die darauf folgende Implementierung vorgestellt. Im Anschluss daran findet sich eine Beschreibung, welche zeigt, wie das Projekt zu starten ist.

Kapitel 2

Architektur

Die App sollte mit dem JavaScript-Framework react-native programmiert werden. Dieses Framework kann eingesetzt werden, wenn die App sowohl auf mobilen Android-, als auch ios-Geräten funktionieren soll. So kann einmal Code geschrieben werden, welche für beide Plattformen verwendet werden kann. Als Programmiersprache wird TypeScript eingesetzt. TypeScript ist wie JavaScript, nur mit Typensicherheit. So können keine Verwirrungen bezüglich des Typs einer Variable aufkommen. Zusammen mit TypeScript und dem Framework react-native wird zusätzlich noch Expo genutzt. Expo ist ein Framework, welches Werkzeuge zur Unterstützung der Entwicklung einer react-native-App bietet.

2.1 Mockups und Komponenten

Um die Anforderungen nicht völlig durcheinander in Quell-Code zu pressen, wurden vorher Prototypen gezeichnet, welche das Design der App darstellen sollen. Dies schafft Struktur und ermöglicht es, systematisch vorzugehen. Ziel der Mockups war es, ein Design zu schaffen, welches möglichst einfach und verständlich alle relevanten Informationen auf einen Blick zeigt und es dem Nutzer ermöglicht, sich ebenfalls weitere Details anzeigen zu lassen, wenn dies gewünscht ist. Anhand dieser Mockups gelang es nun, Komponenten herauszuarbeiten, welche anschließend in Quell-Code umgewandelt werden sollten.

Da die Implementierung mit react-native erfolgt und statt Klassen Komponenten für die Benutzeroberfläche verwendet werden, wurden die Mockups in Komponenten geteilt. Die Komponente, welche alle anderen beinhalten soll, ist die ‚App'. Hier findet sich auch der Button oben rechts, der in Abbildung 2.1(a) zu sehen ist. Eine weitere Komponente bildet die Karte, welche sich über den größten Teil des Bildschirms erstreckt. Klickt man auf einen Marker, der eine Parkmöglichkeit anzeigt, so öffnet man die Komponente der Beschreibung. Diese ist beispielsweise in Abbildung 2.1(b) zu sehen. Hier sollen Kacheln angezeigt werden, welche die einzelnen Informationen liefern. Außerdem soll hier auch der Button angezeigt werden, welcher die Navigation startet.

(a) Dieser Entwurf zeigt die Komponente ‚App' mit dem einem Button oben rechts in der Ecke. Außerdem befindet sich hier auch die Komponente mit der Karte.

(b) Zusätzlich zur Karten-Kompnente in der ‚App' sieht man hier auch noch die Komponente der Beschreibung, welche sich beim Klick auf eine Parkmöglichkeit in der Karte öffnen soll. Die Kacheln zeigen die relevanten Informationen an.

Abbildung 2.1: Diese beiden Entwürfe zeigen jeweils die ‚App'-Komponente und die darin befindliche Karten-Komponente. In der rechten Abbildung öffnet man zusätzlich die Beschreibung-Komponente, welche die Karte verkleinert. (Quelle: Eigens gezeichnete Mockups)

Klickt man auf das Zahnrad in Abbildung 2.1(a) in der Komponente ‚App', so soll sich eine Komponente mit den Einstellungen, wie in Abbildung 2.2(a) zu sehen, öffnen, in welcher der Ton an- und ausgeschaltet werden kann oder auch die Parkmöglichkeiten als Liste angezeigt werden können. Um die Parkmöglichkeiten als Liste zu sehen, wird eine weitere Komponente benötigt, wie sie in Abbildung 2.2(b) erkennbar ist. Die Elemente der Liste werden mit den Favoriten zuerst nach dem Alphabet sortiert.

(a) Die Komponente, welche das Einstellungsmenü beinhaltet. Ein Klick auf den Ton schaltet diesen entweder ein oder aus. Die Kachel darunter zeigt bei einem Klick die Parkmöglichkeiten als Liste an.

(b) Hier befindet sich die Liste der Parkhäuser, welche alphabetisch sortiert ist. Die Favoriten findet man hier ganz oben.

Abbildung 2.2: Auf den Entwürfen ist links das Einstellungsmenü zu sehen. Klickt man auf ‚Parkplätze' oder in der Beschreibung-Komponente auf ‚mehr' so gelangt man zur Ansicht der Details einer Parkmöglichkeit, was auf der rechten Seite zu sehen ist. (Quelle: Eigens gezeichnete Mockups)

Wird in der Liste auf eine Parkmöglichkeit oder in der Beschreibung einer einzelnen auf die Kachel mit der Beschriftung "mehr"geklickt, öffnet sich eine Detailansicht, welche alle vorhandenen Informationen des Parkhauses anzeigt. Zu sehen ist dies in Abbildung 2.3 Diese Übersicht bildet die letzte Komponente.

Abbildung 2.3: Diese Komponente bildet die Detailsansicht der jeweiligen Parkmöglichkeit. Zu erreichen ist diese über die Beschreibung- oder die Einstellung-Komponente. (Quelle: Eigens erstellte Übersicht.)

In Summe sind also zunächst fünf Komponenten geplant, welche in der Komponente ‚App' zu finden sind. Ob dies tatsächlich so umsetzbar ist, wird sich in der Implementierung zeigen.

2.2 Datenbank

Um die Daten nun auch sauber vom Code zu trennen, müssen Datenbanktabellen angelegt werden. Die erste Tabelle ‚parkingarea' wird mit den Daten der Parkmöglichkeiten gefüllt. Hierzu bekommt jedes der neun Parkmöglichkeiten eine ID zur Identifikation. Weitere Spalten in der Tabelle beinhalten:

Name
: Der Name des Parkhauses oder Parkplatzes.

Adresse
: Die Straße oder mehrere Straßen, über die die Parkmöglichkeit angefahren werden kann.

Öffnungszeiten

Wie lange die jeweilige Parkmöglichkeit geöffnet hat.

Preis pro Stunde
Wie viel man pro Stunde bezahlen muss, um sein Fahrzeug zu parken.

Einfahrtshöhe
Wie hoch das Fahrzeug maximal sein darf, um in das Parkhaus zu fahren. Im vorliegenden Fall gibt es auch Parkplätze ohne maximale Einfahrtshöhe.

Favorit
Dieser Wert gibt an, ob es sich um eine favorisiertes Parkmöglichkeit handelt oder nicht. In dem Fall gibt der Wert 1 eine favorisierte Parkmöglichkeit an, wohingegen 0 anzeigt, dass eine Parkmöglichkeit nicht favorisiert wurde.

Latitude
Latitude ist die geographische Breite oder auch der Breitengrad. In Zusammenspiel mit dem Längengrad ergibt sich die exakte Position der Parkmöglichkeit auf der Erde.

Longitude
Die geographische Länge oder auch Längengrad.

In Abbildung 2.4 kann man alle Spalten der Tabelle ‚parkingarea' mit ihren zugehörigen Datentypen erkennen.

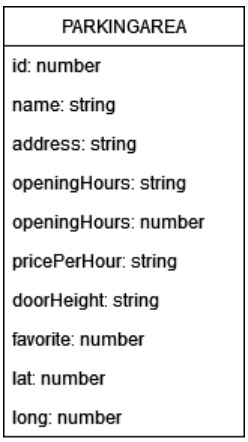

Abbildung 2.4: Die Spalten und ihre Datentypen der Datenbanktabelle ‚parkingarea'. Hier werden in Kapitel 3 alle Parkmöglichkeiten eingetragen. (Quelle: Eigens erstellte Übersicht.)

Die zweite Datenbanktabelle erhielt den Namen ‚parkingareadetails' und sollte mit den Daten der API gefüllt werden. Neben einer einzigartigen ID finden sich hier folgende Spalten:

ID des jeweiligen Parkhauses
Um die Details der Parkmöglichkeiten denen der anderen Tabelle zuordnen zu können, wird zu jedem Datensatz die ID der Parkmöglichkeit gespeichert, zu der die Informationen gehören.

Anzahl der Parkplätze
: Die Anzahl der Gesamtheit aller Parkplätze der jeweiligen Parkmöglichkeit, unabhängig davon, ob diese belegt oder frei sind.

Anzahl belegter Parkplätze
: Die Anzahl der Parkplätze, die aktuell belegt sind.

Anzahl freier Parkplätze
: Die Anzahl der Parkplätze, auf denen noch geparkt werden kann.

Trend
: Der Trend gibt an, ob die Anzahl belegter Parkplätze steigt, sinkt oder gleich bleibt. Wenn beispielsweise viele Menschen auf einmal in die Parkmöglichkeit einfahren, so steigt der Trend. Der Trend kann die Werte 0 für ‚gleichbleibend', 1 für ‚steigend' oder -1 für ‚fallend' annehmen.

Status
: Der Status, in welchem sich die Parkmöglichkeit befindet. Hierfür wird zwischen ‚OK', ‚Ersatzwerte', ‚Manuell' oder ‚Störung' unterschieden.

Geschlossen
: Dieser Wert zeigt an, ob die Parkmöglichkeit aktuell geöffnet oder geschlossen hat. Bei 0 ist die Parkmöglichkeit offen, bei 1 geschlossen.

Datum und Uhrzeit der Daten
: Um die aktuellsten Daten zu verwenden, müssen Datum und Uhrzeit, zu welcher die Daten erstellt sind, gespeichert werden.

In Abbildung 2.5 sieht man nochmals alle Spaltenbezeichnungen und die dazugehörigen Datentypen aufgelistet.

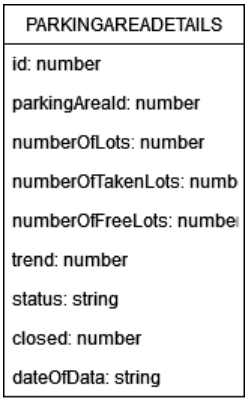

Abbildung 2.5: Um die Daten der API verwenden zu können, wird eine Tabelle ‚parkingareadetails' erstellt. Zu sehen sind hier die Spaltenbezeichnungen und die dazugehörigen Datentypen. (Quelle: Eigens erstellte Übersicht.)

Kapitel 3

Implementierung

In diesem Kapitel wird die Implementierung beschrieben. Zunächst werden die Komponenten und Klassen der Anwendung vorgestellt. Die gezeichneten Mockups unterscheiden sich nur minimal von der tatsächlichen Anwendung. Im Großen und Ganzen ist das Design aber gleich geblieben.

3.1 Komponenten

Die Komponenten bilden alle Funktionen der App, welche visuell auf der Benutzeroberfläche wahrgenommen werden. Neben der Anzeige der Elemente beinhalten sie aber dennoch auch Funktionalität, wie beispielsweise das Aufrufen von Methoden der Datenbank-Klasse ‚DbConnectionService'. Alle Komponenten, mit Ausnahme der ‚App', besitzen ein eigenes Interface. Dieses Interface beinhaltet alle Elemente, die als Properties in die Komponente hineingegeben werden und die jeweils dazugehörigen Typen. Properties können als Parameter an eine Komponente verstanden werden. Dies können neben Strings, Zahlenwerten oder boolschen Werten auch Funktionen oder andere Objekte sein.

3.1.1 Komponente ‚App'

Die ‚App'-Komponente kann sozusagen mit der ‚main()'-Funktion anderer Programmiersprachen verglichen werden. Alles, was in dieser Funktion seinen Platz findet, findet sich auch in der Anwendung selbst wieder. Alle notwendigen Komponenten, die die Anwendung benötigt, werden hier eingefügt.

Zu Beginn werden hier die Tabellen der Datenbank erstellt, sofern diese noch nicht existieren. Diese Komponente verwaltet die Daten. Die Informationen zu den Parkmöglichkeiten, die über die API abgerufen werden, werden zudem auch hier verwaltet. Durch den Hook ‚useEffect' wird sichergestellt, dass die Daten nach einer bestimmten Zeit erneut abgefragt werden. Der Code hierfür kommt aus dem Internet und ist in Listing 3.1 zu sehen.

```
1  useEffect(() => {
2     const intervalCall = setInterval(() => {
3        dbConnectionService.getData();
4     }, MINUTES_MS);
5     return () => {
6        clearInterval(intervalCall);
7     };
8  }, []);
```

Listing 3.1: In diesem Hook werden nach der Zeit, welche sich hinter der Variable ‚MINUTES_MS' verbirgt, die Daten der API abgerufen, was über die Funktion in Zeile 3 geschieht. (Quelle: [1])

Desweiteren gehehn auch die Daten des Eventhooks ‚useGeofenceEvent', welcher in Abschnitt 3.3 erklärt wird, hier ein. Durch die Properties der anderen Komponenten werden die Daten dann an die richtigen Stellen gebracht. Wird ein Geofence betreten, so geschieht auch hier die Sprachausgabe. Der Button für das Ein- und Ausschalten des Tons befindet sich ebenfalls in der ‚App'. Auch die Verwaltung des Tons liegt hier. Wird eine andere Ansicht geöffnet oder der Button zum Stummschalten der Sprachausgabe betätigt, so wird ein ‚useEffect'-Hook getriggert, welcher die Sprachausgabe direkt beendet.

Neben verschiedenen ‚set'- und ‚get'-Funktionen, die für die Weitergabe von Daten zwischen Komponenten verantwortlich sind, befindet sich hier auch das Element, welches benötigt wird, um Toasts auszugeben. Alle Bestandteile in der ‚<View>'-Komponente werden von ‚<RootSiblingParent>' umrahmt, damit nachher an verschiedenen Stellen der Befehl aus Listing 3.2 aufgerufen werden kann und somit in der Anwendung einen Toast anzeigt [2].

```
1  Toast.show(errorMessages.noApiConnectionMessage, {
2     duration: Toast.durations.LONG,
3     position: Toast.positions.BOTTOM,
4  });
```

Listing 3.2: Ein Beispiel des Aufrufs eines Toasts aus der Datei DbConnectionService. (Quelle: Eigene Implementierung)

3.1.2 Komponente ‚ParkingMap'

Die wohl wichtigste Komponente der Anwendung ist die, welche die Karte beinhaltet. Diese findet sich hier. Zunächst wird beschrieben, welche Properties mitgeliedert werden. Hierzu ist es hilfreich, das zugehörige Interface ‚IParkingMap' mit den Typen der Properties zu betrachten:

handleParkingAreaId(id: number): void
Hineingegeben wird eine Funktion ‚handleParkingAreaId', welche nichts zurückgibt. Parameter ist hier ‚id', was den Typ Nummer hat. Diese Funktion bringt die jeweilige ID der Parkmöglichkeit nach draußen in die ‚App' und kann dort

weiterverarbeitet werden.

handleParkingAreaDescription(parkingAreaDescription: boolean): void
‚handleParkingAreaDescription' ist eine Funktion, welche nichts zurückgibt. Der Parameter dieser Funktion ist ein boolscher Wert namens ‚parkingAreaDescription'. Diese Property hat in etwa denselben Nutzen, wie die eben beschriebene. Nur wird hier der ‚App' mitgeteilt, ob die Beschreibung einer Parkmöglichkeit geöffnet werden soll oder nicht.

mapStyle: StyleProp<ViewStyle>
Diese Property ist für das Design der ‚ParkingMap' verantwortlich. Je nachdem, ob die Beschreibung der Parkmöglichkeit angezeigt wird oder nicht, ist die ‚ParkingMap' größer oder kleiner.

Diese Komponete übernimmt die Anzeige der aktuellen Position des Nutzers und das Bilden der Geofences. Damit das funktioniert, benötigt man die Bibliothek Location von expo [3]. Zur Nutzung von Geofences und der Anzeige der aktuellen Position wird die Erlaubnis des Nutzers für das GPS benötigt. Ohne diese Zustimmung wird dem Nutzer eine Meldung angezeigt, dass weder das Geofencing, noch die Anzeige der aktuellen Position funktionieren.

Um nun auch die Karte und die Marker an den Positionen der Parkhäuser anzeigen zu lassen, werden zusätzlich die Komponenten ‚MapView' und ‚Marker' der Bibliothek ‚react-native-maps' benötigt. Je nach Gerät wird dann bei Android Google Maps und bei IOS Apple Maps verwendet. Um nun die Marker an den richtigen Stellen zu plazieren, wird die Liste aller Objekte der Parkmöglichkeiten mit der Funktion ‚map()' durchgegangen und an jeder Stelle der Parkhäuser ein Marker gesetzt. Die Daten der Parkhäuser sind hier nicht aus der Datenbank, sondern aus dem Objekt-Array ‚AllParkingAreas', welches in Abschnitt 3.5 näher erklärt wird. Wird auf einen Marker geklickt, kommen die beiden Funktionen der Properties ins Spiel. Der Aufruf der Funktion ‚handleParkingAreaId' mit der ID der angeklickten Parkmöglichkeit als Parameter lässt die ‚App' wissen, welche Parkmöglichkeit angeklickt wurde. Wenn die Funktion ‚handleParkingAreaDescription' mit dem Parameter ‚true' aufgerufen wird, wird sichergestellt, dass sich dann immer die Beschreibung, welche in Abbildung 3.1 zu sehen ist, öffnet.

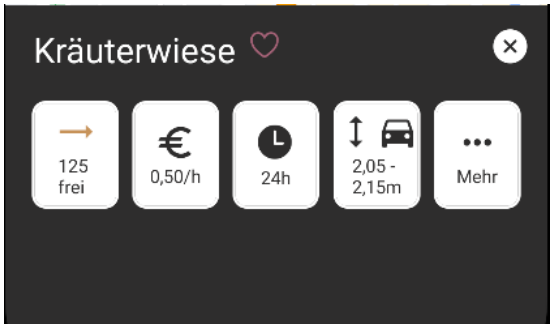

Abbildung 3.1: Die Komponente ‚ParkingAreaDescription', welche bei Klick auf einen Marker aufgerufen wird. (Quelle: Screenshot der erstellten Anwendung.)

3.1.3 Komponente ‚ParkingAreaDescription'

Komponente ‚ParkingAreaDescriptionItemContainer'

3.1.4 Komponente ‚ParkingAreaList'

Komponente ‚ParkingAreaListItem'

3.1.5 Komponente ‚ParkingAreaDetails'

Komponente ‚ParkingAreaDetailsItem'

3.1.6 Komponente ‚ParkingAreaListHeading'

3.2 Datenbankverbindung ‚DbConnectionService'

3.3 Eventhook ‚useGeofenceEvent'

3.4 Models

3.4.1 Interface ‚IParkingArea'

3.4.2 Interface ‚IParkingAreaDetails'

3.4.3 Interface ‚IEventData'

3.5 Parkhausdaten ‚AllParkingAreas'

3.6 Weitere Hilfsmittel

3.6.1 Zentrale Verwaltung der Farben

3.6.2 Zentrale Verwaltung der Strings

Kapitel 4

Starten der App

Literaturverzeichnis

[1] F. Hameed. (2023, Januar 05). How to call an API every minute for a Dashboard in REACT. For those looking for functional components. [online]. Verfügbar: https://stackoverflow.com/questions/48601813/how-to-call-an-api-every-minute-for-a-dashboard-in-react.

[2] Horcrux. (2023, Januar 05). react-native-root-toast. [online]. Verfügbar: https://github.com/magicismight/react-native-root-toast.

[3] Expo. (2023, Januar 05). Location. [online]. Verfügbar: https://docs.expo.dev/versions/v47.0.0/sdk/location/.

[4] Airbnb. (2023, Januar 05). react-native-maps. [online]. Verfügbar: https://github.com/react-native-maps/react-native-maps.

Abbildungsverzeichnis

Anhang A

Anhang