

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

|          |   |           |
|----------|---|-----------|
| <b>1</b> | <b>Einleitung</b>                                   | <b>1</b>  |
| <b>2</b> | <b>Architektur</b>                                  | <b>2</b>  |
| 2.1      | Mockups und Komponenten . . . . .                   | 2         |
| 2.2      | Datenbank . . . . .                                 | 5         |
| <b>3</b> | <b>Implementierung</b>                              | <b>8</b>  |
| 3.1      | Komponenten . . . . .                               | 9         |
| 3.1.1    | Komponente ‚App‘ . . . . .                          | 9         |
| 3.1.2    | Komponente ‚ParkingMap‘ . . . . .                   | 10        |
| 3.1.3    | Komponente ‚ParkingAreaDescription‘ . . . . .       | 11        |
| 3.1.4    | Komponente ‚ParkingAreaList‘ . . . . .              | 13        |
| 3.1.5    | Komponente ‚ParkingAreaDetails‘ . . . . .           | 15        |
| 3.1.6    | Komponente ‚ParkingAreaListHeading‘ . . . . .       | 16        |
| 3.2      | Datenbankverbindung ‚DbConnectionService‘ . . . . . | 17        |
| 3.3      | Eventhook ‚useGeofenceEvent‘ . . . . .              | 17        |
| 3.4      | Models . . . . .                                    | 17        |
| 3.4.1    | Interface ‚IParkingArea‘ . . . . .                  | 17        |
| 3.4.2    | Interface ‚IParkingAreaDetails‘ . . . . .           | 17        |
| 3.4.3    | Interface ‚IEventData‘ . . . . .                    | 17        |
| 3.5      | Parkhausdaten ‚AllParkingAreas‘ . . . . .           | 17        |
| 3.6      | Weitere Hilfsmittel . . . . .                       | 17        |
| 3.6.1    | Zentrale Verwaltung der Farben . . . . .            | 17        |
| 3.6.2    | Zentrale Verwaltung der Strings . . . . .           | 17        |
| <b>4</b> | <b>Starten der App</b>                              | <b>18</b> |
|          | <b>Literaturverzeichnis</b>                         | <b>19</b> |
|          | <b>Abbildungsverzeichnis</b>                        | <b>20</b> |
| <b>A</b> | <b>Anhang</b>                                       | <b>21</b> |

# Kapitel 1

## Einleitung

Im Rahmen der Studienarbeit sollte eine App als Parkleitsystem für die neun Parkhäuser um den Altstadttring 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

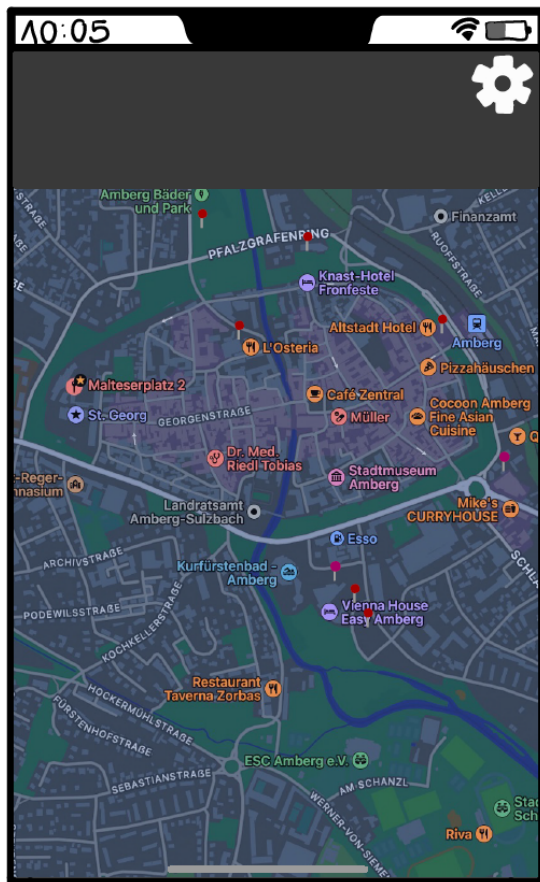
## Architektur

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.

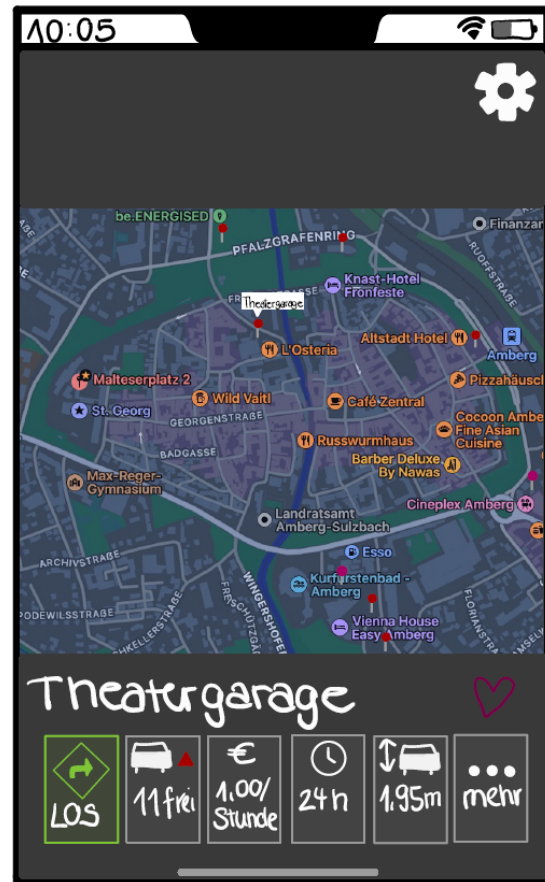
### 2.1 Mockups und Komponenten

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-Komponente 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.

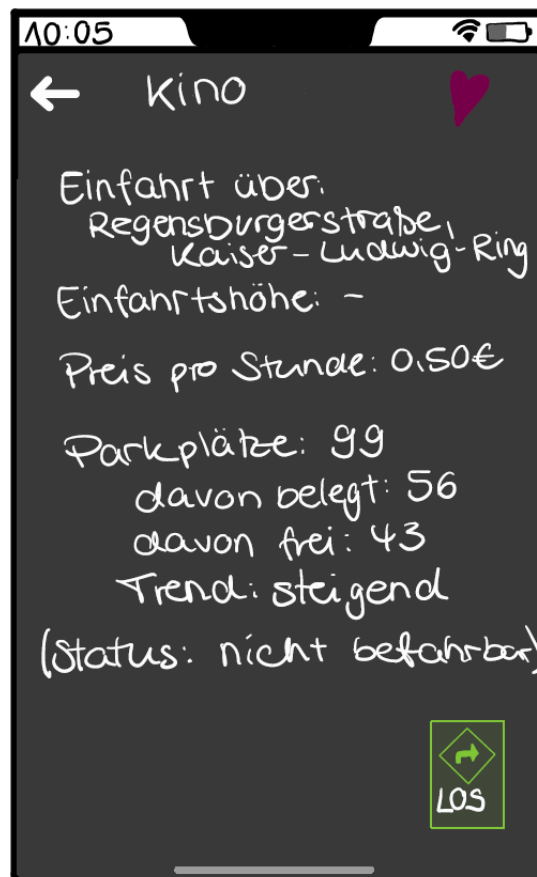


(a) Die Komponente, welche das Einstellungsmenü beinhaltet. Ein Klick auf den Ton schaltet diesen entweder ein oder aus. Die Kachel darunter zeigt bei einem Klick die Parkmöglichkeiten als Liste an.

(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 Einstellungskomponente. (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 favorisierte 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.

| PARKINGAREA          |
|----------------------|
| id: number           |
| name: string         |
| address: string      |
| openingHours: string |
| openingHours: number |
| pricePerHour: string |
| doorHeight: string   |
| favorite: number     |
| lat: number          |
| long: number         |

**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.

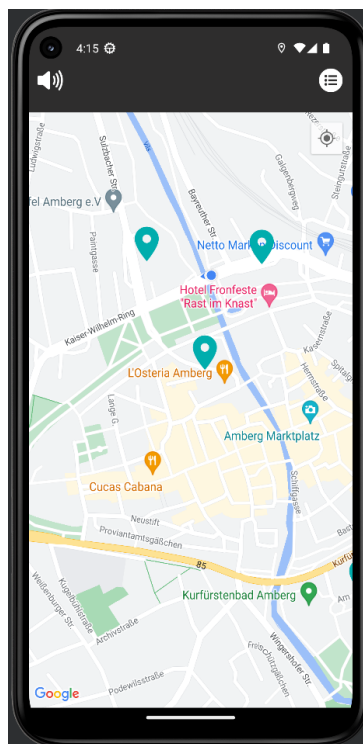
| PARKINGAREADETAILS      |
|-------------------------|
| id: number              |
| parkingAreaid: number   |
| numberOfLots: number    |
| numberOfTakenLots: numb |
| numberOfFreeLots: numbe |
| trend: number           |
| status: string          |
| closed: number          |
| dateOfData: string      |

**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, was bei Betrachtung von Abbildung 3.1 auffällt.



**Abbildung 3.1:** Die Startseite der fertigen Anwendung, auf welcher man nun die Parkmöglichkeiten sieht und durch Klick auf einen Marker nähere Informationen findet. Oben rechts kann auch eine Liste der Parkmöglichkeiten aufgerufen werden. In der linken oberen Ecke kann der Ton an- und ausgeschaltet werden. (Quelle: Screenshot der erstellten Anwendung.)

## 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 gehen 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 mitgeliefert 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 boolescher 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.

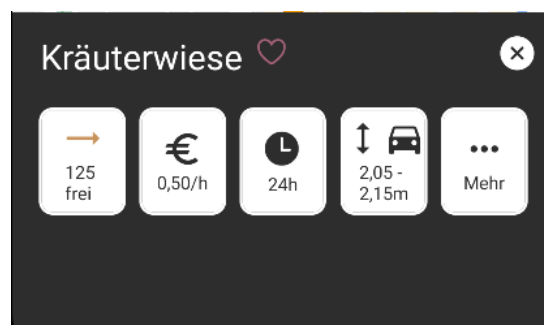
#### **mapStyle: StyleProp<ViewStyle>**

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 Komponente 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 platzieren, 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 3.2 zu sehen ist, öffnet.



**Abbildung 3.2:** Die Komponente ‚ParkingAreaDescription‘, welche bei Klick auf einen Marker aufgerufen wird. (Quelle: Screenshot der erstellten Anwendung.)

### 3.1.3 Komponente ‚ParkingAreaDescription‘

In dieser Komponente werden die relevantesten Daten einer Parkmöglichkeit in Kacheln angezeigt, wie in Abbildung 3.2 zu sehen ist. Auch für die Properties dieser Komponente gibt es ein Interface ‚IParkingAreaDescription‘:

#### **dbConnectionService: DbConnectionService**

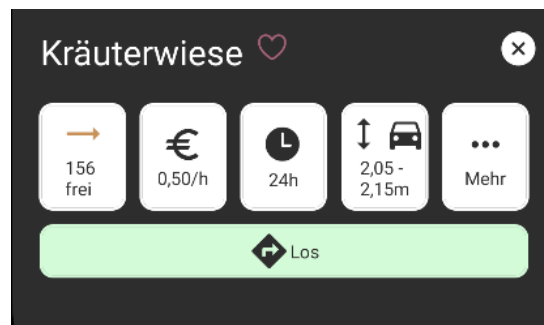
Dies ist eine Instanz des ‚DbConnectionService‘, welche in der ‚App‘ am Anfang erstellt wird. Diese Property ist für die Datenbankverbindung zuständig.

#### **id: number**

Mit der Nummer ID kommt die ID der angeklickten Parkmöglichkeit in die Beschreibung. Nun kann mit Hilfe von ‚dbConnectionService‘ in der Datenbank nach den passenden Daten gesucht werden.

#### **geofenceEventData: IEventData[]**

Die Daten des Events ‚useGeofenceEvent‘ werden in einem Objekt mit dem Interface IEventData gespeichert. Ein Array aus diesen Objekten bekommt auch die ‚ParkingAreaDescription‘, damit unterhalb der Informationskacheln ein Button für die Navigation angezeigt werden kann, falls der Nutzer sich im Geofence einer Parkmöglichkeit befindet. Dieser Button ist auch auf Abbildung 3.3 zu sehen.



**Abbildung 3.3:** Die Komponente ‚ParkingAreaDescription‘, welche bei Klick auf einen Marker aufgerufen wird. (Quelle: Screenshot der erstellten Anwendung.)

**handleShowParkingAreaDescription(parkingAreaDescription: boolean): void**

Diese Funktion vom Typ ‚void‘ mit der dazugehörigen boolschen Variable lässt die Beschreibung schließen. Wird dieser Wert auf ‚false‘ gesetzt, so ist diese Funktion dafür zuständig, dies der App mitzuteilen, welche wiederum für die Anzeige dieser Beschreibung zuständig ist. Es wird geprüft, ob ‚parkingAreaDescription‘ ‚true‘ oder ‚false‘ anzeigt. Bei ‚true‘ öffnet sich diese Komponente, bei ‚false‘ wird sie geschlossen. Dies geschieht beispielsweise, wenn der kleine weiße Kreis in der rechten Ecke in Abbildung 3.3 betätigt wird.

**handleParkingAreaDetails(parkingAreaDetails: boolean): void**

Der boolsche Parameter der Funktion zeigt an, ob auf den ‚Mehr‘-Button gedrückt wird. Ist der Button gedrückt worden, so wird die Komponente ‚ParkingAreaDetails‘ geöffnet.

**handleParkingAreaData(parkingAreaData: IParkingArea): void**

Diese Funktion beinhaltet ein Objekt des Typs ‚IParkingArea‘, was das Interface für einen Eintrag einer Parkmöglichkeit ist und weiter unten beschrieben wird. Die Daten zur angeklickten Parkmöglichkeit gelangen so zur App.

**handleParkingAreaDetailData(parkingAreaDetails: IParkingAreaDetails): void**

Diese Property macht genau das gleiche, wie die vorherige: Es werden Daten zu App getragen. In dem Fall handelt es sich um die Daten, wie voll die Parkmöglichkeit zum aktuellen Zeitpunkt ist.

**handleDataBaseError(databaseError: boolean): void**

Diese Funktion beinhaltet einen boolschen Parameter, welcher der App mitteilt, wenn es zu einem Fehler in der Datenbank gekommen ist.

**handleVolume(volume: boolean): void**

Öffnet man die Navigation mit dem ‚Los‘-Button oder die Details mit dem ‚Mehr‘-Button, so wird der App mitgeteilt, dass der Ton der Sprachausgabe beendet werden soll. Wird dies weggelassen, so wird die Sprachausgabe fortgesetzt, wenn man möglicherweise schon gar nicht mehr in der App, sondern schon bei der Navigation ist.

In der ‚ParkingAreaDescription‘-Komponente werden die Daten der angeklickten Parkmöglichkeit aus beiden Datenbanken abgerufen und dann in Kacheln dargestellt.

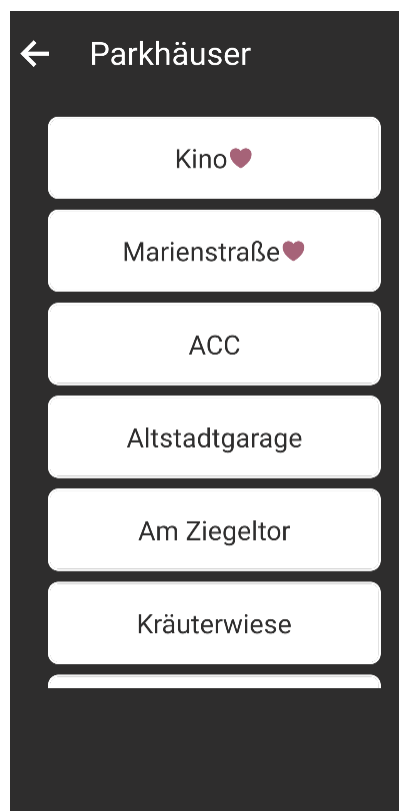
Um den Code übersichtlicher zu gestalten, sind die Kacheln eine weitere Komponente: ‚ParkingAreaDescriptionItemContainer‘, welche in Abschnitt 3.1.3 näher beschrieben wird. Zudem kann hier das Parkhaus favorisiert werden, was direkt in die Datenbank eingetragen wird. Kommt es zu einem Datenbankfehler, so wird dies dem Nutzer über ein Alert mitgeteilt. Damit die Abfrage der Datenbank klappt und die Daten vor dem Rendern in den einzelnen Komponenten der ‚ParkingAreaDescription‘-Komponente stehen, muss die Abfrage asynchron ablaufen. Das heißt, dass gewartet werden muss, bis die Daten da sind, bevor sie angezeigt werden.

### Komponente ‚ParkingAreaDescriptionItemContainer‘

Diese Komponente ist die Kachel der ‚ParkingAreaDescription‘. Auch sie hat ein Interface für die Properties. Dieses beinhaltet aber nur den Text, einen oder zwei Namen für Icons und die Farbe der Icons. Ansonsten besitzt diese Komponente keinerlei Funktionalität. Sie ist nur für das Anzeigen zuständig und, dass in der ‚ParkingAreaDescription‘-Komponente weniger redundanter Code ist.

### 3.1.4 Komponente ‚ParkingAreaList‘

In Abbildung 3.4 ist diese Komponente zu sehen. Hier sind die Parkmöglichkeiten mit Favoriten zuerst als anklickbare Kacheln alphabetisch aufgelistet.



**Abbildung 3.4:** Die Komponente ‚ParkingAreaList‘, zu welcher man über einen Klick auf das Icon oben rechts in der App kommt. Die Liste ist mit Favoriten zuerst alphabetisch geordnet. (Quelle: Screenshot der erstellten Anwendung.)

Die ‚ParkingAreaList‘-Komponente besitzt auch ein Interface für ihre Properties. So können die Typen dieser sichergestellt werden. Das Interface sieht folgendermaßen aus:

**dbConnectionService: DbConnectionService**

Diese Property vom Typ DbConnectionService ist wichtig für die Abfrage der Daten aus der Datenbank.

**handleShowParkingAreaList(showParkingAreaList: boolean): void**

Diese Funktion ist dafür zuständig, die ‚ParkingAreaList‘-Komponente zu schließen, wenn man auf den Zurück-Pfeil drückt. In Abbildung 3.4 ist die Liste mit dem Zurück-Pfeil zu sehen.

**handleParkingAreaDescription(parkingAreaDescription: boolean): void**

Um nicht die Beschreibung ohne Daten zu haben, wenn die ‚ParkingAreaDescription‘-Komponente geöffnet ist, wenn man in die ‚ParkingAreaList‘-Komponente geht und wieder zurückkommt, wird die Beschreibung geschlossen, sobald der Zurück-Pfeil gedrückt wird.

**handleParkingAreaDetails(parkingAreaDetails: boolean): void**

Diese Funktion erfüllt denselben Zweck wie ‚handleParkingAreaDetails‘ in Unterabschnitt 3.1.3.

**handleParkingAreaData(parkingAreaData: IParkingArea): void**

Auch hier tut die Funktion das Gleiche, wie ‚handleParkingAreaData‘ in Unterabschnitt 3.1.3.

**handleParkingAreaDetailData(parkingAreaDetails: IParkingAreaDetails): void**

In Unterabschnitt 3.1.3 unter ‚handleParkingAreaDetailData‘ ist beschrieben, was auch diese Funktion macht.

**handleDataBaseError(databaseError: boolean): void**

Die Beschreibung dieser Property befindet sich in Unterabschnitt 3.1.3 unter ‚handleDataBaseError‘.

Die Überschrift dieser Komponente geht aus der Komponente ‚ParkingAreaListHeadig‘ hervor, welche in Unterabschnitt 3.1.6 zu finden ist. Ist bei der Datenbank-Abfrage in dieser Komponente kein Fehler aufgetreten, so werden die Namen der Parkmöglichkeiten als Flatlist dargestellt. Die Kacheln, in welchen sich die Namen der Parkmöglichkeiten befinden, gehen aus der Komponente ‚ParkingAreaListItem‘ hervor. Klickt man auf eine dieser Kacheln, so kommt es zu einer weiteren Abfrage der Datenbank, wobei ‚handleParkingAreaData‘ und ‚handleParkingAreaDetailsData‘ der eben beschriebenen Properties mit den Daten gefüllt werden. Die Property ‚handleParkingAreaDetails‘ wird dann auf ‚true‘ gesetzt, um die Komponente in Unterabschnitt 3.1.5 zu öffnen. Je nachdem, ob ein Fehler bei der Datenbankabfrage auftritt, wird der Parameter in ‚handleDataBaseError‘ auf ‚true‘ oder ‚false‘ gesetzt.

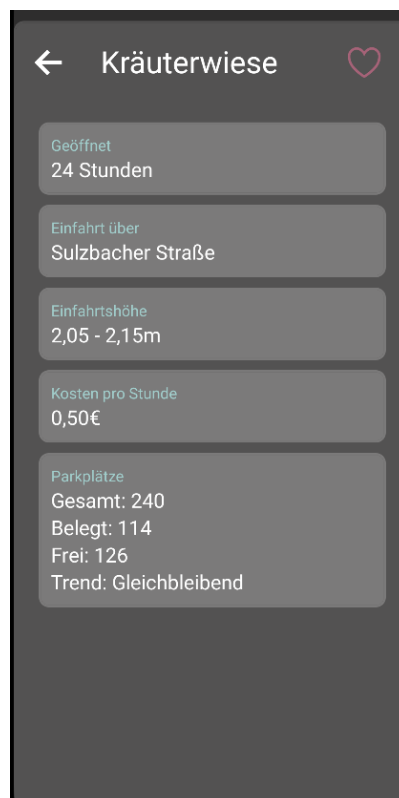


## Komponente ‚ParkingAreaListItem‘

Diese Komponente besitzt, wie alle anderen auch ein Interface. Da es sich um anklickbare Kacheln handelt, muss man dieser Komponente die Funktion mitteilen, welche beim Klick ausgeführt werden soll. Für die Ausführung der Funktion braucht es zudem noch die ID der Parkmöglichkeit. Um den Namen und, falls die Parkmöglichkeit favorisiert wurde, ein Herz anzuzeigen, müssen auch noch der anzuzeigende Name als String und ein numerischer Wert für die Favorisierung mitgegeben werden. 0 heißt, dass die Parkmöglichkeit nicht unter die Favoriten fällt. 1 bedeutet, dass diese favorisiert wurde.

### 3.1.5 Komponente ‚ParkingAreaDetails‘

Wie bereits beschrieben, öffnet sich diese Komponente beim Klick auf ‚mehr‘ in Abbildung 3.2 oder bei Auswahl einer Parkmöglichkeit in Abbildung 3.4. Diese Komponente dient lediglich zum Anzeigen von Daten. Dennoch kann auch hier eine Parkmöglichkeit zu den Favoriten hinzugefügt oder entfernt werden. Zu sehen ist diese Ansicht in Abbildung 3.5.



**Abbildung 3.5:** Die Details zu einer Parkmöglichkeit. Hier werden alle vorhandenen Daten zur Parkmöglichkeit angezeigt. Sollte diese geschlossen sein oder aus anderen Gründen nicht befahrbar sein, wird dies dem Nutzer mit einem Text in einer roten Kachel mitgeteilt. Der Nutzer kann auch hier Parkmöglichkeiten favorisieren oder von den Favoriten entfernen. (Quelle: Screenshot der erstellten Anwendung.)

Die Properties, welche durch das Interface ihren Typ erhalten, sind folgende:

**dbConnectionService: DbConnectionService**

Diese Property ist dafür zuständig, dass beim Drücken auf das Herz in der oberen rechten Ecke eine Parkmöglichkeit in der Datenbank als Favorit oder kein Favorit eingetragen werden kann.

**handleShowParkingAreaDetails(parkingAreaDetails: boolean): void**

Diese Funktion wird benötigt, wenn der Pfeil in der oberen linken Ecke gedrückt wird. Tut der Nutzer das, wo wird ‚parkingAreaDetails‘ auf ‚false‘ gesetzt und die Ansicht aus Abbildung 3.5 wird geschlossen.

**parkingAreaData: IParkingArea**

Das Objekt, welches Parkhausdaten, wie die Einfahrtshöhe oder den Preis, beinhaltet.

**parkingAreaDetailsData: IParkingAreaDetails**

Das Objekt, welches die Daten der API beinhaltet, welche unter ‚Parkplätze‘ in Abbildung 3.5 dargestellt sind.

**databaseError: boolean**

Ist diese Property ‚true‘, so bekommt der Nutzer eine rot gefärbte Kachel mit einem Text angezeigt, der ihn darauf hinweist, dass die Daten nicht angezeigt werden können. In diesem Fall ist die Aktion zu wiederholen.

Da hier die Kacheln ebenfalls aus jeweils den gleichen ‚View‘-Komponenten und demselben Design bestehen, wurde hier wieder eine Komponente ‚ParkingAreaDetailsItem‘ erstellt, welche sich darum kümmert, die Elemente schön anzuzeigen. Die Überschrift kommt ebenfalls durch die Komponente ‚ParkingAreaListHeading‘ in Unterabschnitt 3.1.6 zustande, wie die Überschrift in Unterabschnitt 3.1.4.

**Komponente ‚ParkingAreaDetailsItem‘**

Diese Komponente ist ebenfalls nur zum Anzeigen von Dateien. Ihre Properties sind der Text, der in der Kachel jeweils oben kleiner angezeigt wird und jener Text, welcher etwas größer darin steht und anhängig von der angeklickten Parkmöglichkeit ist. Außerdem wird eine boolesche Variable hineingegeben, die dieser Komponente mitteilt, ob es sich um die Anzeige eines Fehlers handelt. Ist das der Fall, werden die Kacheln rot gefärbt.

**3.1.6 Komponente ‚ParkingAreaListHeading‘**

Diese Komponente beinhaltet das Design der Überschrift, wie in Abbildung 3.4 oder Abbildung 3.5 zu sehen. Als Property werden hier lediglich die Funktion, die beim Klick auf den Zurück-Pfeil erfolgen soll und der Text, welcher dargestellt werden soll, hineingegeben.

### **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

|     |   |    |
|-----|---|----|
| 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. . . . .  | 3  |
| 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. . . . .   | 4  |
| 2.3 | Diese Komponente bildet die Detailsansicht der jeweiligen Parkmöglichkeit. Zu erreichen ist diese über die Beschreibung- oder die Einstellung-Komponente. . . . .   | 5  |
| 2.4 | Die Spalten und ihre Datentypen der Datenbanktabelle ‚parkingarea‘. Hier werden in Kapitel 3 alle Parkmöglichkeiten eingetragen. . . . .  | 6  |
| 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. . . . .  | 7  |
| 3.1 | Die Startseite der fertigen Anwendung, auf welcher man nun die Parkmöglichkeiten sieht und durch Klick auf einen Marker nähere Informationen findet. Oben rechts kann auch eine Liste der Parkmöglichkeiten aufgerufen werden. In der linken oberen Ecke kann der Ton an- und ausgeschaltet werden. . . . .   | 8  |
| 3.2 | Die Komponente ‚ParkingAreaDescription‘, welche bei Klick auf einen Marker aufgerufen wird. . . . .   | 11 |
| 3.3 | Die Komponente ‚ParkingAreaDescription‘, welche bei Klick auf einen Marker aufgerufen wird. . . . .   | 12 |
| 3.4 | Die Komponente ‚ParkingAreaList‘, zu welcher man über einen Klick auf das Icon oben rechts in der App kommt. Die Liste ist mit Favoriten zuerst alphabetisch geordnet. . . . .  | 13 |
| 3.5 | Die Details zu einer Parkmöglichkeit. Hier werden alle vorhandenen Daten zur Parkmöglichkeit angezeigt. Sollte diese geschlossen sein oder aus anderen Gründen nicht befahrbar sein, wird dies dem Nutzer mit einem Text in einer roten Kachel mitgeteilt. Der Nutzer kann auch hier Parkmöglichkeiten favorisieren oder von den Favoriten entfernen. . . . | 15 |

# Anhang A

## Anhang