

Diplomarbeit



SkateBuddy

Eine App für Skater

erstellt von

Philipp Schuler
Alexander Bertoni
Maximilian Neuner



HTBLuVA
Innsbruck Anichstrasse

Betreuer:
Sabo Rubner

2021/22

Eidesstattliche Erklärung

Ich erkläre hiermit an Eides statt durch meine eigenhändige Unterschrift, dass ich die vorliegende Diplomarbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe. Alle Stellen, die wörtlich oder inhaltlich den angegebenen Quellen entnommen wurden, sind als solche kenntlich gemacht.

Die vorliegende Arbeit wurde bisher in gleicher oder ähnlicher Form noch nicht als Diplomarbeit eingereicht.

Innsbruck, am 22. März 2022

Verfasser/Verfasserinnen:

Philipp Schuler

Alexander Bertoni

Maximilian Neuner

Projektteam



Philipp Schuler

Adresse

PLZ Ort

Tel: -

E-Mail: pschuler@tsn.at



Alexander Bertoni

Adresse

PLZ Ort

Tel: -

E-Mail: abertoni@tsn.at



Maximilian Neuner

Adresse

PLZ Ort

Tel: -

E-Mail: maximilianeuner@tsn.at

Betreuer



Bakk. Sabo Rubner

HTBLuVA Innsbruck Anichstraße

E-Mail: sabo.rubner@htlinn.ac.at

Danksagung

Hans-Jürgen Vogt für die Idee. Nicolaus und Joshi für Planung, myPHPAdmin, Server Hosting,
Logo, Docker, Motivation Adrian für Name SkateBuddy Adi für Testung

Philipp Schuler & Alexander Bertoni & Maximilian Neuner

Innsbruck, 22. März 2022

Gendererklärung

Aus Gründen der besseren Lesbarkeit wird in dieser Diplomarbeit durchwegs die Sprachform des generischen Maskulinums verwendet. An dieser Stelle wird darauf hingewiesen, dass die ausschließliche Verwendung der männlichen Sprachform geschlechtsunabhängig gewertet werden soll.

Abstract

When a person decides to try skateboarding for themselves, often times, they quit before they have even learned their first trick. Our app wants to reveal the vast number of skateparks Tyrol has to offer, so that new skaters can try them out and maybe find new friends there, motivating them to keep going, which will support the local skateboarding community.

Zusammenfassung

Neue Skater haben oft keine Auskunft über die lokalen Parks bzw. wissen gar nicht wie viele Skateparks es mittlerweile in Tirol gibt.

In der App wollen wir alle umliegenden Parks auflisten und eine kleine Übersicht bieten (Hindernisse mit Schwierigkeitsstufen, Skateshops in der Nähe, Bewertungen der Benutzer, usw.). Außerdem ist es den Usern möglich Videos ihrer Tricks hochzuladen und zu teilen.

Inhaltsverzeichnis

I. Intro	12
1. Meta	13
1.1. Problem / Vorgeschichte	13
1.2. Projektlogo	13
2. Aufbau	14
2.1. Erste Planung	14
2.2. Komponenten	14
II. Theoretische Grundlagen	15
3. Webentwicklung	16
3.1. Website	16
3.2. HTML	16
3.2.1. Was ist HTML?	16
3.2.2. Die Geschichte von HTML	16
3.3. CSS	17
3.3.1. Was ist CSS?	17
3.3.2. Die Geschichte von CSS	17
3.4. JavaScript	17
3.4.1. First-Class-Functions	18
3.4.2. Objektorientierte Programmierung	19
3.4.3. React	19
4. Mobile App Entwicklung	24
4.1. Apps	24
4.1.1. Native Apps	24
4.1.2. Webanwendungen	25
4.1.3. Hybrid-Apps	25
4.1.4. Cross-Platform-Apps	25
4.2. Android: Entwicklung für die OpenSource-Welt	26
4.2.1. Was ist Android?	26
4.2.2. Geschichte	26
4.2.3. Verwendung	26
4.2.4. App-Entwicklung	27
4.3. iOS: Entwicklung für das Apple-Ökosystem	28
4.3.1. Was ist iOS?	28
4.3.2. Geschichte	28
4.3.3. App-Entwicklung	28

Inhaltsverzeichnis

4.4. React Native	29
4.4.1. Was ist React Native?	29
4.4.2. Wie viel React steckt wirklich in React Native?	29
4.4.3. Geschichte	29
4.4.4. Wer benutzt React Native?	30
4.4.5. Lizenz	30
4.5. Alternative Kandidaten	31
4.5.1. Flutter	31
4.5.2. Xamarin	31
5. Backend	32
5.1. Server	32
5.1.1. Was ist ein Server?	32
5.1.2. Node.js	34
5.1.3. NodeJs Anwendungsbereich	35
5.2. Datenbank	35
5.2.1. Definition	35
5.2.2. Datenbankmanagementsystem	35
5.2.3. Structured Query Language (SQL)	36
5.2.4. MySQL	36
5.2.5. Andere Datenbanktypen	37
5.2.6. Data Warehouse	37
5.2.7. NoSQL-Datenbanken	37
5.2.8. Diagrammdatenbanken	38
5.2.9. OLTP-Datenbanken	38
5.3. Docker	40
5.3.1. Was ist Docker?	40
5.3.2. Wie genau funktioniert Docker?	40
5.3.3. Unterschied Docker Container und herkömmliche Linux-Container	40
5.3.4. Vorteile von Docker	40
III. Webseite	42
6. Allgemeines	43
6.1. Idee	43
6.2. Daten bekommen	44
6.3. Daten senden	45
6.4. Benutzertoken	46
6.4.1. Token dekodieren	46
6.4.2. Token Überprüfung	46
6.4.3. Token entfernen	46
6.5. Index.js	47
6.6. App.js	47
6.6.1. React Router, Routes, Route	47
6.6.2. Navigationsleiste	48
6.6.3. Fußzeile	49
6.7. Slideshow	50

Inhaltsverzeichnis

6.8.	Map	51
6.8.1.	Marker	52
6.9.	Design	52
7.	Views	54
7.1.	Startseite	54
7.2.	Account erstellen	54
7.3.	Login	57
7.3.1.	Augensymbol	58
7.4.	Profil	59
7.5.	Liste der Parks	60
7.5.1.	Suchleiste	61
7.6.	Vorschläge erstellen	61
7.7.	Admin-Page	62
7.7.1.	Recommendation-View	62
7.7.2.	Park hinzufügen	63
7.8.	Park Details	63
7.8.1.	Bewertungen	65
IV.	Mobile App	66
8.	Vorbereitung	67
8.1.	Warum habe ich mich für React Native entschieden?	67
9.	Erstellung des Projekts	68
9.1.	React Native CLI	69
9.1.1.	Abhängigkeiten und Erstellung des Projekts	69
9.1.2.	Ordnerstruktur	70
9.1.3.	Metro	74
9.2.	Expo CLI	77
9.2.1.	Installation und Erstellung einer Expo App	77
10.	Aufbau	81
10.1.	Screens	81
10.2.	Arten von Screens	81
10.2.1.	Listen	81
10.2.2.	Details	81
10.2.3.	Formulare	81
10.3.	Komponenten	82
10.3.1.	Snippets	82
10.3.2.	Text	83
10.3.3.	Button	84
10.3.4.	TextInput	85
10.4.	Styles	86
10.4.1.	Schriftart	86
10.4.2.	Farbwerte	86
10.4.3.	Icons	86
10.4.4.	FlashMessage	87

Inhaltsverzeichnis

11. Hooks	88
11.1. useFetch	88
11.2. useDirections	89
11.2.1. Api-Key	89
11.3. useLocation	90
12. Authentifizierung	94
12.1. AuthContext	95
12.1.1. state & dispatch	95
12.1.2. authContext	97
12.1.3. AuthProvider	98
12.2. AuthHandler	100
13. Navigation	102
13.1. Tab-Navigation	103
13.1.1. MapScreen	104
13.1.2. ProfileScreen	108
13.2. Stack-Navigation	109
13.2.1. SkateparksStack	109
13.2.2. LoginSignupStack	116
V. Backend	120
14. Allgemeines	121
14.1. Datenbankverbindung	121
14.2. Rest-API	121
14.2.1. HTTP / HTTP's	121
14.2.2. Authorization	122
15. Rest-Routes	125
15.1. Route-Routes	125
VI. Appendix	154
Zeitaufwand	155
Literaturverzeichnis	161
Abbildungsverzeichnis	164
Code-Snippet-Verzeichnis	166

Teil I.

Intro

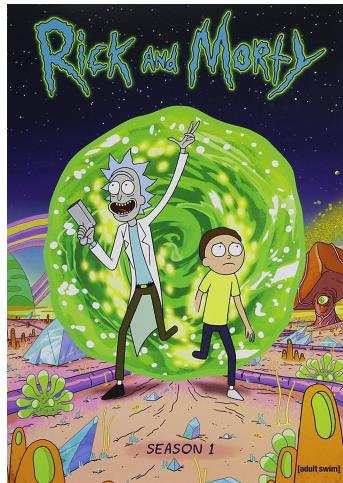
1. Meta

1.1. Problem / Vorgeschichte

Hier soll geklärt werden: Warum machen wir genau diese Diplomarbeit, wie sind wir aufs Thema gekommen, was ist die Problemstellung, was ist unsere Lösung, was sind andere Möglichkeiten?

1.2. Projektlogo

Auf dem Logo ist ein Skateboard zu sehen, welches durch ein Portal aus einem Smartphone fliegt. Der Designer des Logos ließ sich vom Portal aus der Serie *Rick and Morty* inspirieren und wollte damit die Verbindung zwischen der Technik und dem Sport zum Ausdruck bringen.



(a) Rick and Morty Staffel 1
Werbebild [1]



(b) SkateBuddy Projektlogo

Abbildung 1.1.: Vergleich zwischen Portal aus der Serie und dem Logo

2. Aufbau

2.1. Erste Planung

Bei den anfänglichen Überlegungen stand schnell fest, dass wir uns auf zukunftsorientierte und bewährte Technologien stützen wollen.

2.2. Komponenten

Grafische Darstellung

Teil II.

Theoretische Grundlagen

3. Webentwicklung

3.1. Website

Eine Website ist ein Dokument im HTML-Format, welches sich durch einen Browser anzeigen lässt. Jede Website besitzt eine URL (Uniform Resource Locator) mit welcher sie erreichbar ist. Die Start- bzw. Hauptseite besitzt hier meist die URL *www.website.com*. Seitenabschnitte welche auf ein bestimmtes Verzeichnis hinweisen, haben am Ende der URL meist den Namen des Verzeichnisses stehen. Diese nennt man *Landingpages*. So eine URL würde dann wie folgt aussehen → *www.website.com/Verzeichnis*. Das Gesamte, also mit allen Landingpages gemeinsam wird dann als **Website** bezeichnet. Auf einer Website befindet sich nicht nur HTML, sondern auch die Gestaltungssprache CSS. Diese ist dafür zuständig dem HTML-Dokument ein anschauliches Aussehen zu geben. [2]

3.2. HTML

3.2.1. Was ist HTML?

HTML steht für *Hypertext Markup Language* und ist der grundlegende Baustein um eine Internetanwendung zu erstellen. HTML ist eine Beschreibungssprache, mit der sich die logischen Strukturen eines Dokuments beschreiben lassen. Solche wären zum Beispiel: Kapitel, Unterkapitel, Absätze und eingebundene Bilder. Dafür verwendet man verschiedene von HTML zur Verfügung gestellte Befehle. Der Browser verarbeitet den geschriebenen Text sowie die Befehle und wandelt diese in eine grafische Oberfläche um.

3.2.2. Die Geschichte von HTML

Da das Internet immer populärer wurde und somit immer mehr Daten im Internet veröffentlicht wurden, brauchte man allgemeine Regeln um ein Chaos zu vermeiden. Das Ziel von HTML war es, Dokumente und Daten unabhängig von der eingesetzten Hard- und Software anzeigen zu lassen. Die erste HTML-Version war für die Strukturierung und Darstellung von wissenschaftlichen Informationen gedacht und wurde 1989 von *Tim Berners-Lee* entwickelt. Die Hypertext Markup Language wird ständig weiterentwickelt und befindet sich im Moment in der Version 5.2. [3]

HTML hat sich in den letzten Jahren wie folgt entwickelt:

3. Webentwicklung

Jahr	Version	Neues Feature
1993	HTML	Text und Bildgeneration
1995	HTML 2.0	Formulare
1997	HTML 3.2	Tabellen, Applets
1997	HTML 4.0	CSS
2014	HTML 5	Neues Vokabular
2017	HTML 5.2	Neue Features

3.3. CSS

3.3.1. Was ist CSS?

Mit CSS (*Cascading Style Sheets*) kann das Aussehen von HTML-Dokumenten bestimmt werden. CSS beeinflusst also um keiner Weise den Inhalt der Seite, sondern nur das Design. Der Anwender kann mit CSS also die Schriftarten, Farbe, Höhen und Breiten einer Website definieren.

3.3.2. Die Geschichte von CSS

1994 wurde CSS erstmals von *Håkon Wium Lie* vorgeschlagen. Zu dieser Zeit arbeitete *Bert Bos* an ein Darstellungsprogramm names *Argo*, welcher dabei seine eigene Stilvorlagensprache benutzte. Die beiden taten sich letztendlich zusammen und entwickelten CSS. CSS war jedoch nicht die einzige Sprache, welche das Ziel hatte die Darstellung von HTML Dokumenten zu verändern. Sie war jedoch die erste Sprache, welche Regeln definierte, die über mehrere Stilvorlagen hinweg vererbt werden konnten. Das W3C (W3C ist das Gremium zur Standardisierung der Techniken im World Wide Web) wurde erstmals auf CSS aufmerksam bei einer Präsentation von Håkon und Bos an der „Mosaic and the Web“ Konferenz in Chicago 1995. Sie arbeiteten zusammen mit anderen Mitgliedern weiter an CSS und am Dezember 1996 wurde die *CSS Level 1 Recommendation* publiziert. Viele aktuell gängige Darstellungsprogramme benutzen zurzeit CSS 2.1. Diese Version wurde am 7. Juni endgültig vom W3C empfohlen. Voraussetzung für diese Empfehlung war, dass es pro Merkmal mindestens zwei Programme gibt, welche es korrekt interpretieren können.

CSS Level 3 ist derzeit in Entwicklung und ist im Gegensatz zu den Vorgängern modular aufgebaut. Dies bedeutet, dass einzelne Teiltechniken so wie die Steuerung von Sprachausgaben eigene Versionsschritte und Entwicklungszeiten besitzen. Manche Module wurden bereits als fertig empfohlen und werden auch schon in Darstellungsprogrammen implementiert. [4]

3.4. JavaScript

JavaScript ist eine Programmiersprache/Skriptsprache, die üblicherweise in Webseiten Verwendung findet. Sie wird jedoch auch noch in vielen anderen Umgebungen außerhalb der Website Entwicklung benutzt, wie zum Beispiel Node.js. JavaScript besitzt First-Class-Functions (Funktionen erster Klasse). Außerdem ist JS eine prototypbasierte Sprache, welche mehreren Paradigmen folgt und dynamisch als sowohl auch objektorientiert ist. JavaScript ist plattformunabhängig. Zusätzlich ist es sehr kompakt und ressourcenschonend. JavaScript sollte man

3. Webentwicklung

nicht mit Java verwechseln. Sie besitzen beide eine unterschiedliche Syntax, Semantik und Verwendung. [5]

3.4.1. First-Class-Functions

Funktionen erster Klasse werden wie Variablen behandelt. Bei einer Programmiersprache welche First-Class-Funktionen besitzt, kann man Funktionen als Parameter übergeben, einer Variable zuweisen oder von einer anderen Funktion zurückgegeben werden. [6]

3.4.1.1. Beispiel für die Zuweisung einer Funktion an eine Variable

```
1 const v = function() {
2     Console.log("Ausgaben in der Konsole")
3 }
4
5 v();
```

Hier wird der Variable *v* eine anonyme Funktion zugewiesen, welche in der Konsole eine Ausgabe liefert. Diese Funktion wird dann ganz einfach über die Variable mittels den zwei Klammern aufgerufen.

3.4.1.2. Beispiel für das Übergeben einer Funktion als Argument

```
1 function sagHallo() {
2     return "Hallo, ";
3 }
4 function gruessen(gruss, name) {
5     Console.log(gruss(), name);
6 }
7
8 gruessen(sagHallo(), "JavaScript!");
```

Hier übergeben wir der Funktion *gruessen* 2 Parameter. Einer dieser beiden Parameter ist eine Funktion namens *sagHallo*, welche "Hallo, " als return liefert. Führen wir nun die Methode *gruessen* mit diesen Parametern aus, wird in die Console der Satz "Hallo, Javascript" geschrieben.

3.4.1.3. Eine Funktion als Return

```
1 function sagHallo(){
2     return function(){
3         Console.log("Hallo")
4     }
5 }
```

Hier wird ganz einfach als return-Wert eine Funktion übergeben.

3. Webentwicklung

3.4.2. Objektorientierte Programmierung

Objektorientierte Programmierung (OOP) eignet sich gut für große und komplexe Software, welche aktiv aktualisiert oder gewartet werden muss. OOP konzentriert sich nicht auf die Logik, sondern auf die Objekte, mit denen das Programm interagieren soll. Der Aufbau für das Programm bei der Objektorientierten Entwicklung ist auch für das Entwickeln von Vorteil. Andere Vorteile von OOP sind die Wiederverwendbarkeit von Code, Skalierbarkeit und die Effizienz. [7]

3.4.2.1. Prinzipien von OOP

Verkapselung

Objekte werden privat innerhalb einer definierten Grenze oder Klasse gehalten und implementiert. Andere Objekte haben keinen Zugriff auf diese Klasse und keine Berechtigung Änderungen vorzunehmen. Dies versichert eine größere Programmiersicherheit.

Abstraktion

Objekte verbergen den unnötigen Implementierungscode und offenbaren nur interne Mechanismen, welche für andere Objekte relevant sind. Hilfreich für Entwickler um Änderungen und Ergänzungen vorzunehmen.

Inheritance

Man kann Beziehungen und Unterklassen zwischen Objekten zuweisen, um gemeinsame Logik wiederverwenden zu können. Vorteile hiervon sind eine verkürzte Entwicklungszeit und es sorgt für eine höhere Genauigkeit.

3.4.2.2. Prototypbasierte Programmierung

Ist eine andere Art von objektorientierter Programmierung, bei der keine Klassen verwendet werden. Stattdessen erzeugt man Objekte durch das Kopieren von bereits existierenden Objekten (Prototypen). Beim Kopieren werden alle Eigenschaften des kopierten Objekts übernommen. Man kann diese auch verändern und/oder ergänzen.

3.4.3. React

React beschreibt sich selbst als JavaScript Bibliothek und nicht wie andere als ein Framework. React ist viel mehr eine Anleitung wie man verschieden Dinge machen könnte. React ist also kein All-in-One Paket.

React wurde entwickelt um eine möglichst performante Oberfläche zu gestalten. Dabei werden einfache Views erstellt und React übernimmt die Änderung der Daten innerhalb dieser Views. React erstellt einen sogenannten virtuellen DOM bei der Datenanzeige.

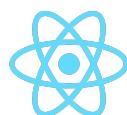


Abbildung 3.1.: ReactJS Logo [8]

3. Webentwicklung

3.4.3.1. Virtuelles DOM

VDOM ist ein Programmierkonzept, bei dem durch den Prozess *Reconciliation* eine ideale, virtuelle Darstellung der Benutzeroberfläche im Speicher gehalten wird und mit dem realen DOM, durch eine Bibliothek wie ReactDOM, synchronisiert wird. Dies ermöglicht eine deklarative API von React, bei welcher man den Zustand der Benutzeroberfläche angibt und React stellt sicher, dass das DOM diesem Zustand entspricht. Dies entfernt verschiedene Arbeitsschritte, welche sonst durchgeführt werden müssten. VDOM ist viel mehr ein Muster als eine spezifische Technologie. Der Begriff virtuelles DOM wird in React oft mit Objekten in Verbindung gebracht, da diese die Benutzeroberfläche darstellen. Um jedoch zusätzliche Information über den Komponentenbaum zu speichern, benutzt React interne Objekte nämlich sogenannte *Fiber*.[9]

3.4.3.2. Single-Page-App

Wenn sich bei einer Webanwendung die View ändert, wird dies oft mit einem Seitenwechsel assoziiert. Dies ist zwar nicht falsch, da frühere Webanwendung in der Tat solch einen Seitenwechsel durchführten. Dies war jedoch sehr ineffizient, da bei einem Wechsel der View immer eine neue Seite geladen werden musste. Eine modernere Herangehensweise an diese Sache sind sogenannte *Single-Page-Apps (SPA)*. Sie bestehen aus nur einer Seite mit mehreren Views. Anstatt dass bei einem View Wechsel eine neue Seite geladen wird, änderst sich je nach State die aktive Seite. [10]

3.4.3.3. States

Die Stärke in React liegt in den States. Ein State ist der Istzustand der Anwendung. Ändert sich die View der Seite, ändert sich auch der State. Sobald sich irgendetwas auf der Seite ändert, ändert sich der State der Seite. Ändert sich der State innerhalb einer Komponente, wird die Renderfunktion erneut ausgeführt. React wird mit der Funktion *setState()* darüber informiert, dass die Renderfunktion erneut ausgeführt werden muss.

3. Webentwicklung

3.4.3.4. Die Renderfunktion

In der Renderfunktion wird ein JSX-Objekt zurückgegeben, welches dann wie üblich in HTML übersetzt wird.

```
1 class RenderBeispiel{
2     render{
3         return(
4             <div>
5                 <p>Ich bin ein Beispiel fuer eine Renderfunktion</p>
6             </div>
7         );
8     }
9 }
```

In diesem Beispiel gibt die Renderfunktion ein `<p>`-Element mit dem Text **Ich bin ein Beispiel für eine Renderfunktion** aus.

Ich bin ein Beispiel für eine Renderfunktion

Abbildung 3.2.: Ergebnis Renderfunktion

So würde die Renderfunktion von oben dann auf der Website aussehen.

3.4.3.5. Komponenten

In React bestehen die Views aus sogenannten Komponenten. Mit React können vordefinierte CodeBits programmiert werden, welche am Ende zu einer View zusammengesetzt werden. Eine React Anwendung besteht also aus logisch getrennten Einheiten, welche unabhängig voneinander behandelt werden. Bei React wird kein Pattern mit MVC oder MVVM verwendet. In React wird Logik als sowohl auch Template in der selben JS-Datei verwaltet. HTML-Code wird in React in einer JavaScript Datei geschrieben und zwar im sogenannten *JSX-Format*. JSX ist HTML jedoch sehr ähnlich.[11]

In React gibt es zwei verschiedene Arten eine Komponente zu erstellen.

- Komponenten mit State
- Komponenten ohne State (Funktionale Komponenten)

Komponenten mit State

Sie besitzen einen im Konstruktor initialisierten State. Sie sind immer Klassenkomponenten.

Komponenten ohne State

Zum Erstellen von Komponenten ohne State können entweder Funktionen oder Klassen verwendet werden. Stateless Funktionskomponenten sind vom großen Vorteil und sollten immer verwendet werden, es sei denn, es wird ein Lebenszyklus-Hook benötigt. Stateless Komponenten sind viel einfacher in der Verwendung.[12]

3. Webentwicklung

3.4.3.6. JSX

JSX ist eine Erweiterung der Syntax von JavaScript. Es bestimmt wie die Benutzeroberfläche aussehen soll und erinnert sehr an eine Vorlagensprache. JSX kommt jedoch mit der Kraft von JavaScript. React basiert grundlegend darauf, dass die Renderlogik mit der Benutzeroberflächenlogik verbunden wird. JSX ist für React nicht notwendig, jedoch ist es ein sehr hilfreiches Werkzeug. Außerdem ist es sehr hilfreich um Fehler zu finden.

```
1 function example_jsx(){
2     const j = "Hallo, ich bin JavaScript code"
3
4     return(
5         <div>
6             <p>{ j }</p>
7         </div>
8     )
9 }
```

Innerhalb der geschwungenen Klammern, kann jede beliebige Art von JavaScript-Code ausgeführt werden. In diesem Falle würde das `<p>` Element *Hallo, ich bin JavaScript code* anzeigen. [13]

3.4.3.7. Props

Damit die Komponenten in React miteinander kommunizieren können, werden Props benötigt. Props ist ein spezielles Schlüsselwort, welches für Eigenschaften steht und welches für die Übergabe von Daten einer Komponente an eine andere verwendet wird. Daten von Props sollen nur in einer Richtung weitergegeben werden und zwar vom Elternteil zum Kind. Egal wie eine Komponente deklariert wird, ihre Props dürfen nie verändert werden.

Eine Funktion welche nicht ihre eigenen Eingaben ändert und bei selben Eingaben immer das selben Ergebnis liefert wird *pure* genannt.

```
1 function sum(a,b){
2     return a + b;
3 }
```

In diesem Beispiel einer *pure Function* ist sichtbar, dass die Eingabewerte nie geändert werden und egal wie oft die selben Eingabewerte übergeben werden, die Funktion wird immer das selbe Ergebnis liefern.

Das Gegenteil einer “pure Function” ist eine *impure Function* eine Funktion, welche ihre Eingaben ändert.

```
1 function withdraw(account, amount) {
2     account.total -= amount;
3 }
```

[14]

3. Webentwicklung

3.4.3.8. Vor- und Nachteile von React

Vorteile

- Daten und Präsentationen sind getrennt
- Einfacher Einstieg
- Verhält sich wegen virtuellen DOM immer flüssig

Nachteile

- Benötigt zusätzliche Bibliotheken zum Erstellen einer vollen Website
- Lohnt sich nur bei interaktiven Webseiten

[15]

3.4.3.9. React hooks

Hooks wurden in der React Version 16.8 hinzugefügt. Sie sind eine Erweiterung von React und sind zu 100% mit bestehenden Code kompatibel. Mit ihnen kann man States und andere React-Funktionen außerhalb von Klassenkomponenten benutzen. Durch diese Hooks werden Komponenten unabhängiger.

3.4.3.10. Alternativen zu React

- Inferno JS
- React-Lite
- Aurelia
- Riot JS
- Vue JS
- Cycle.js
- Preact
- Backbone JS
- Ember JS
- Mithril
- AngularJS

In dieser Diplomschrift wird jedoch nicht genauer auf die einzelnen Alternativen eingegangen, da sie nicht in der Diplomarbeit verwendet wurden.

4. Mobile App Entwicklung

4.1. Apps

Als Apps (von engl. Applications), zu deutsch Anwendungen, werden Softwarepakete für jede Art von Computer bezeichnet. Sie erfüllen meistens einen bestimmten Zweck und können von Anwendern auf dem gewünschten Gerät installiert werden.

Computer-Anwendungen werden in fast allen Branchen der Wirtschaft eingesetzt, um Menschen bei der Arbeit zu unterstützen. Apps für Unterhaltung und Social Media sind wohl die Apps, welche am meisten Bildschirmzeit von Anwendern beanspruchen.

Von einfachen ToDo-Listen-Apps bis zu komplexen Buchhaltungssystemen kann praktisch alles realisiert werden, wofür die Leistung des Geräts reicht.

Die Umsetzung einer App-Idee kann durch mehrere Herangehensweisen angegangen werden.

4.1.1. Native Apps

Die Applikation wird direkt auf das Zielbetriebssystem aufgebaut und hat tiefen Zugriff auf die Software- und Hardwareschnittstellen des Geräts. Es gibt jedoch meist viele Wege, also Sprachen und Frameworks, um auf der Zielplattform die vorgegebene App zu erstellen.

- Windows: C, C++, C#
- MacOS und iOS: Swift, Objective-C
- Linux: C
- Android: Kotlin, Java, C++

4.1.1.1. Vorteil

Ein großer Vorteil davon ist die Performance. Es gibt keine Zwischen-Ebene, mit der kommuniziert werden muss, sondern es wird die in das System eingebaute Render-Engine verwendet.

4.1.1.2. Nachteil

Die meisten Firmen wollen ihre Anwendungen auf mehreren Plattformen anbieten, bei einer Smartphone-App also sowohl eine Android-App im Google Play Store, als auch eine iOS-App im App Store. Um beides mit Native Apps zu realisieren benötigt man also zwei App-Entwicklungs-Teams, die zwei verschiedene Repositories pflegen müssen und welche beide gleichzeitig neue Features einbauen und testen müssen.

4. Mobile App Entwicklung

4.1.2. Webanwendungen

Eine mögliche Lösung ist die Webanwendung. Alles, was dafür auf dem Endgerät vorhanden sein muss, ist ein Webbrowser, der die Webseite darstellen kann. Diese Webseite muss nun nur noch von einem einzigen Team entwickelt und an unterschiedliche Bildschirmgrößen angepasst werden.

Durch die Verwendung einer Web-App im Browser geht jedoch ein großer Teil des Zugriffs auf die gerätespezifische API verloren. Außerdem funktioniert eine Webanwendung nur mit einer aktiven Internetverbindung und es muss bei jedem Screen-Wechsel ein neuer Web-Request verarbeitet werden.

4.1.3. Hybrid-Apps

In einer Hybrid-App werden Native App und Webanwendung kombiniert. Die Native App zeigt in einer sogenannten WebView die gewünschte Webseite an und hat gleichzeitig auch Zugriff auf Software- und Hardware-Schnittstellen. Der Endbenutzer sollte dabei nicht mitbekommen, dass die Anwendung eigentlich nur einen Webbrowser darstellt. Beispiel: Ionic Framework.

4.1.4. Cross-Platform-Apps

Eine Cross-Platform-App hat genau wie eine Hybrid-App Zugriff auf tiefere Ebenen der Geräte-API, der Unterschied besteht nur darin, dass die Benutzeroberfläche von einer einheitlichen Auszeichnungssprache (Markup Language) wie XML bzw. XAML oder JSX in die jeweiligen UI-Elemente der Zielplattform umgewandelt werden. Beispiele für Frameworks dieser Art sind: React Native, Xamarin.

4. Mobile App Entwicklung

4.2. Android: Entwicklung für die OpenSource-Welt

4.2.1. Was ist Android?

Das Betriebssystem *Android* basiert auf dem Linux-Kernel und ist seit 2012 das am meisten verwendete Betriebssystem für Smartphones mit einem Marktanteil von etwa 70% weltweit [16]. Die aktuelle Version ist Android 12, sie wurde am 4. Oktober 2021 veröffentlicht.

Android selbst ist freie Software [17], welche dadurch definiert ist, dass jeder Anwender die Software für jeden Zweck verwenden, alle Teile des Quellcodes ändern und das Ergebnis kopieren und verteilen darf. Jedoch werden die meisten Android-Mobilgeräte mit vorinstallierter, nicht-freier Software von den Geräte-Herstellern ausgestattet, was die eigentliche Installation auf dem Gerät proprietär macht.

4.2.2. Geschichte

Im Jahre 2008 veröffentlichte *Google LLC* die erste Version des Betriebssystems, welches zuvor von Andrew Rubin für die Steuerung von Digitalkameras entwickelt wurde. Es sollte in den nachfolgenden Jahren von der *Open Handset Alliance*, einem Unternehmenszusammenschluss aus Firmen im IT-Sektor, weiterentwickelt werden und offene Standards für Mobilgeräte festlegen [18]. Es werden seitdem jedes Jahr neue Android-Versionen veröffentlicht, die neueste Version ist Android 12. Frühere Versionen verwendeten Süßigkeiten als Codenamen:

- Android 4.1: Jelly Bean
- Android 4.4: KitKat
- Android 5: Lollipop
- Android 6: Marshmallow
- Android 7: Nougat
- Android 8: Oreo
- Android 9: Pie

Diese Codenamen steigen alphabetisch, daher wird Android 10, obwohl es keinen echten Codenamen mehr trägt, auch manchmal mit einem (Q) geschrieben.

4.2.3. Verwendung

Google stellt selbst eine Reihe von Mobilgeräten her, darunter bis 2015 das Nexus und heutzutage das Google Pixel, mit dem Google Pixel 6 Pro als Flaggenschiff von 2021. Die Smartphones sind mit einer Basis-Version von Android ausgestattet, wohingegen die Tablets und Notebooks Chrome OS verwenden.

Viele Geräte-Hersteller entwickeln und pflegen eine eigene Abwandlung des Android Open Source Projekts, sogenannte Aufsätze. Dabei werden meistens Elemente der Benutzeroberfläche stark, jedoch der grundsätzliche Aufbau gar nicht bis minimal verändert. Bekannte Android-Aufsätze sind:

4. Mobile App Entwicklung

- One UI von Samsung (Südkorea)
- MIUI von Xiaomi (China)
- EMUI von Huawei (China)
- OxygenOS von OnePlus (China)

4.2.4. App-Entwicklung

Für Android wird von Google die Entwicklungsumgebung Android Studio zur Verfügung gestellt, welche auf IntelliJ IDEA von Jetbrains aufbaut und alle benötigten Werkzeuge, für das Entwickeln von Apps bereitstellt. Außerdem werden Microsoft Windows, Apple MacOS und Linux als Entwicklungsplattform unterstützt.

Die Programmierung der Anwendung erfolgt hauptsächlich mittels Java und XML. Alternativ zu Java kann auch Kotlin, eine Programmiersprache, welche extra für die Erstellung von Smartphone-Apps optimiert wurde, benutzt werden.

Das Basiselement für die Entwicklung ist das Android Software Development Kit (SDK), zu deutsch Android Softwareentwicklungspaket. Dieses kann durch die offizielle Entwicklungsumgebung Android Studio heruntergeladen und installiert werden.

4. Mobile App Entwicklung

4.3. iOS: Entwicklung für das Apple-Ökosystem

4.3.1. Was ist iOS?

iOS wird das Betriebssystem genannt, welches von Apple exklusiv für mobile Geräte entwickelt wird, hauptsächlich für die iPhone-Produktreihe. Die aktuelle Version ist iOS 15.4, sie wurde am 20. September 2021 veröffentlicht.

4.3.2. Geschichte

Die erste Version von iOS wurde im iPhone 1 im Jahr 2007 gezeigt. Das erste iPhone, welches am 9. Januar 2007 von Steve Jobs, dem damaligen CEO von Apple, vorgestellt wurde, war die wohl einflussreichste Erfindung des 21. Jahrhunderts. Es kombinierte die drei Funktionen Multimedia, Telefonie und Internet in einem Gerät, welches noch dazu einfach zu bedienen war und keine physische Tastatur benötigte, im Gegensatz zu vorherigen Smartphones wie das Motorola Q, BlackBerry oder Nokia E62. Die Bedienung des Touch-Screens erfolgt auch nicht mit einem Eingabestift, sondern per Hand. So gesehen war es das erste Mobiltelefon, welches wirklich den Namen Smartphone verdient hatte.

Das iPhone war sofort ein voller Erfolg und machte Apple zu einem der erfolgreichsten Unternehmen der Welt. Nach der Veröffentlichung veränderte sich die Smartphone-Industrie drastisch, viele Hersteller setzten im Design auf weniger Knöpfe und mehr Bildschirmfläche.

4.3.3. App-Entwicklung

Um Apps für das Apple-Ökosystem zu entwickeln, benötigt man natürlich einen PC, auf welchem MacOS installiert ist. Die Programmierung erfolgt hauptsächlich in der Hauseigenen IDE von Apple, XCode, und den Sprachen Objective-C und, seit 2014, auch mittels Swift.

4.4. React Native

4.4.1. Was ist React Native?

React Native ist ein JavaScript-Framework, welches dafür verwendet wird, Cross-Platform-Apps für die Betriebssysteme Android, iOS, Windows und MacOS zu entwickeln. Außerdem ist der Zugriff auf die native Plattform-API möglich.

4.4.2. Wie viel React steckt wirklich in React Native?

Wie der Name schon vermuten lässt baut React Native hauptsächlich auf die JavaScript-Bibliothek React auf. Sie hilft bei der Erstellung der Benutzeroberfläche, verwaltet den Zustand der Anwendung und hat noch viele weitere kleine Aufgaben. React an sich ist noch eine Bibliothek, mit – absichtlich – wenig "Grundgerüst". Das macht es auch für die Open-Source-Gemeinschaft einfacher zusammen gute Lösungen zu entwickeln und so werden die wenigen wichtigen Aufgaben von React dafür perfekt ausgeführt. Detaillierte Informationen zu State, Render-Funktionen, Komponenten und JSX finden Sie im Kapitel React.

4.4.3. Geschichte

Die ersten Versionen der Facebook-App für Smartphones war eine Hybrid-App, also eine Webseite, welche einfach in einer nativen App eingebunden wird [19]. Die Leistung der Handy-App war für das Niveau des Social-Media-Giganten eindeutig zu schlecht, die App musste neu geschrieben werden in Objective-C, womit Facebook sich eine 2,5-fach schnellere Anwendung erwartete [20].

Nachdem React veröffentlicht worden war, versuchten die Entwickler bei Facebook einen effizienten Weg zu finden, die neue Technologie zu nutzen um native Anwendungen, genauer eigentlich Cross-Platform Anwendungen, zu schreiben.

2015 wurde die erste stabile Version von React Native veröffentlicht. Seit Anfang an verwendet Facebook intern in ihren erfolgreichsten Apps React Native als Basistechnologie. Die Firma Microsoft stellt selbst Bibliotheken zur Verfügung, um React Native Anwendungen auch für die Universal Windows Platform (UWP) und MacOS zu entwickeln.

4. Mobile App Entwicklung

4.4.4. Wer benutzt React Native?

Laut der eigenen Webseite von React Native benutzen einige der größten Firmen der Welt React Native als Cross-Platform-Framework für ihre Android- und iOS-Apps.

Unternehmen	App	Android	iOS
Facebook	Facebook Facebook Ads Manager Facebook Analytics Instagram Oculus		☒
Microsoft	Skype		☒
Discord	Discord	☐	☒
Tesla	Tesla		☒
Coinbase	Coinbase		☒
Walmart	Walmart		☒
Pinterest	Pinterest		☒
Uber	Uber Eats		☒
Shopify	Shopify		☒
Wix.com	Wix.com		☒

Berühmte Beispiele für Apps mit React Native [21]

Discord ist das einzige Unternehmen in der Liste, welches nur für die Plattform iOS React Native verwendet. In einem Blog Post erklären sie, dass React Native auf Android ihrer Meinung nach keine akzeptable Performance in Sachen Reaktionsfähigkeit von Knöpfen liefert [22]. Dieser Post ist aus dem Jahr 2018 und seitdem hat sich React Native stark verbessert. Im selben Jahr begann nämlich auch die Entwicklung von Fabric, der neuen Render-Engine von React Native, welche 2021 dann schließlich in die offizielle Facebook App übernommen wurde.

4.4.5. Lizenz

React Native wird unter der MIT-Lizenz geführt, das heißt dass jeder die Software sowohl für Open-Source als auch für Closed-Source-Projekte verwenden darf.

4. Mobile App Entwicklung

4.5. Alternative Kandidaten

4.5.1. Flutter

Flutter ist ein Framework, welches von Google erstellt wurde und heutzutage zu den modernsten seiner Art gehört. Es benutzt für die Benutzeroberflächenerstellung die neuartige Sprache Dart, die extra für dieses Framework entwickelt wurde und kann für die Erstellung von Cross-Platform-Applikationen eingesetzt werden. Es unterstützt die Zielplattformen Android, iOS, Windows, MacOS, Linux, Web und sogar Google Fuchsia [23].

4.5.2. Xamarin

Das Cross-Plattform-Framework Xamarin ist, im Vergleich mit Xamarin und React Native, schon am Längsten in Entwicklung. Es wird hauptsächlich in den Sprachen C# und XAML verfasst und ist auch für praktisch alle Zielplattformen geeignet. Jedoch gibt es einen entscheidenden Nachteil: Für jede Zielplattform gibt es eine eigene Version des Frameworks. Es teilt sich auf in XamarinNative und Xamarin.Forms, letzteres unterstützt die Cross-Plattform Eigenschaften.

2020 wurde von Microsoft verkündet, dass sie eine Fusion aus Xamarin und .NET 6 planen, um ein neues Framework zu erschaffen. Dieses soll den Namen .NET MAUI (.NET Multi-Platform App UI) tragen [24].

5. Backend

5.1. Server

5.1.1. Was ist ein Server?

Als *Server* eng(Diener) wird sowohl ein Computer mit einem Netzwerk genannt, als auch ein Programm, das auf diesen Pc läuft. Somit gibt es also **zwei** Definitionen für einen Server.

5.1.1.1. Server (Hardware)

Als Hardware Server wird ein Rechnernetz aus einem Computer, auf dem neben dem Betriebssystem mehrere softwarebasierte Server laufen. Bezeichnet wird ein solcher hardwarebasierter Server als **Host** (englisch für Gastgeber). Jeder Rechner kann im Prinzip als Host verwendet werden.

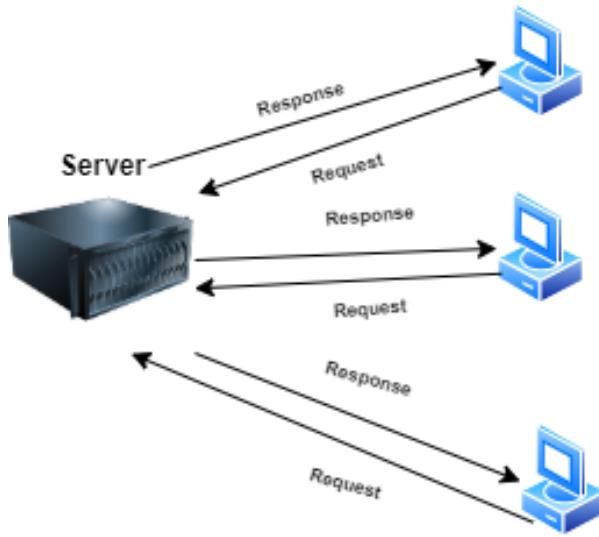
5.1.1.2. Server (Software)

Ein software Server ist in der Regel ein Programm oder ein Anwendung der von anderen Rechnern den sogenannten Clients lokal über ein Netzwerk in Anspruch genommen werden kann. Die Art der Server der Server Software bestimmt den Dienst des Servers. Grundlage für eine Kommunikation bildet das Client-Server-Modell. 5.2.9.1

5.1.1.3. Funktionsweise eines Servers

Wie bereits erwähnt basiert die Kommunikation zwischen den Server und den Clients auf der Grundlage des *Client-Server-Modells*. Jeder Dienst der, im Netzwerk wird von einem dauerhaft erreichbaren Server zur Verfügung gestellt. Somit kann sichergestellt werden, dass Dienste wie Webbrower E-Mail zu jeder Zeit auf den Server zugreifen können um den Dienst in Beanspruchung nehmen zu können.

5. Backend



5.1.1.4. Welche Arten von Servern gibt es?

Webserver Ein Webserver hat die Aufgabe, Websites zu speichern, aufzubereiten und an die Webbrower bzw. Clients auszugeben. Kommuniziert wird zwischen Server und Software mittels **Hypertext Transfer Protocol (HTTP)** bzw mittels der verschlüsselten Variante **HTTPS**. Die gängigsten Webserver sind Apache HTTP Server, Microsoft Internet Information Services (IIS) oder Nginx.

File-Server Ein File Server hat die Aufgabe zur zentralen Speicherung von Dateien, welche von den Clients über ein Netzwerk zugänglich gemacht werden. Ein File Server ermöglicht aufgrund von verschiedenster Dateiversionen Konflikte entgegenzuwirken. Außerdem erfolgt eine automatische Versionierung der Daten sowie ein zentrales Back-up sämtlicher Daten. Falls der Zugriff auf den Server über das Web läuft kommen Protokolle wie **FTP** (File Transport Protocol), **SFTP**(Secure File Transfer Protocol), **FTPS**(FTP over SSL) oder **SCP**(Secure Copy) zum Einsatz. Im lokalen Gebrauch sind die Protokolle **SMB**(Server MEssage Block) und **NFS**(Network File System).

Mailserver Wie des Name es bereits andeutet ermöglicht ein Mail Server E-Mails zu empfangen, zu senden und weiterzuleiten. Er besteht aus mehreren Software-Modulen. Zum Einsatz kommt hier in der Regel das **Simple Mail Transfer Protocol**(SMTP). User die einen Mail-Server nutzen wollen benötigen einen E-Mail Client, der die Nachrichten vom Server holt und im E-Mail-Postfach darstellt. Ein solcher Abruf erfolgt über **IMAP**(Internet Message Access Protocol) oder **POD** (Post Office Protocol).

Datenbank-Server Ein Datenbank-Server auch genannt **Backend-Server** ermöglicht es Programme, über ein Netzwerk, zugriff auf ein oder mehrere Datenbanksysteme zu gewährleisten. Die wohl bekanntesten Software-Lösungen sind *Oracle*, *MySql*, *Microsoft SQL Server*, *PostgreSQL* und *DB2*. Um mit dem Backend Server kommunizieren zu können benötigt der User das sogenannte **Frontend**.

5. Backend

Proxy-Server Ein Proxy-Server dient als Kommunikationsschnittstelle in einem Rechnernetzwerk. Er nimmt Anfragen aus dem Netzwerk entgegen und leitet diese über seine IP-Adresse weiter. Eingesetzt wird ein solcher Proxy-Server um Kommunikationen zu filtern, Bandbreite zu kontrollieren, Verfügbarkeit durch Lastverteilung zu erhöhen um Daten zwischenzuspeichern auch **Caching** genannt. Mithilfe eines Proxy-Servers ist es ebenfalls möglich Anonymität zu gewährleisten, da die IP-Adressen der Clients hinter dem Proxy verborgen bleiben.

Game-Server Ein Game-Server ist ein Server der speziell für onlinebasierte Multiplayer-Spiele entwickelt wurde. Die Daten des Online Spiels werden verwaltet und somit ist eine synchrone Interaktion mit der virtuellen Welt möglich. Die Hardware für einen solchen Server kann in einem Rechenzentrum oder lokal bereit gestellt werden.

DNS-Server DNS-Server haben die Aufgabe als Namensauflöser in einem Netzwerk zu dienen. Sie sind also dazu zuständig das sie Hostename wie www.skate-buddy.com in die entsprechende IP-Adresse übersetzen.

5.1.1.5. Server Hosting

Da das Anschaffen der Server Hardware durchaus teuer werden kann lohnt es sich für Privatpersonen, die ein eigenes Server Projekt realisieren möchten auf gemietete Ressourcen zurückzugreifen. Verschiedensten Hosting-Modelle werden von spezialisierten Provider angeboten, dadurch müssen sich die Benutzer nicht mehr um den Betrieb der Hardware zu kümmern.

[25]

5.1.2. Node.js

Node.js ist eine serverseitige, eventbasierte Laufzeitumgebung für die Programmiersprache *JavaScript* (*JavaScript*), mit der es möglich ist *JavaScript* Code außerhalb des Webbrowsers auszuführen.

Sie wurde 2009 von Ryan Dahl entwickelt als Anwendung, für die Entwicklung skalierbarer Netzwerkanwendungen.

NodeJS basiert auf der *V8-Engine* also auf der selben Engine wie *Google Chrome*. Das heißt also, dass NodeJS die selbe Engine zum Ausführen von JavaScript-Code verwendet. wie *Google Chrome* Die *V8-Engine* ist eine prozessbasierte virtuelle Maschine, die mittels eines Just-In-Time (JIT) Compilers, den geschriebenen JavaScript Code, zur Laufzeit in Maschinencode übersetzt.

Obwohl *NodeJS* hauptsächlich für die Programmierung von, Webserver Programme eingesetzt wird, kann es auch für die Entwicklung von Skripten oder Tools oder bei Entwicklung von Desktop- oder Echtzeitanwendungen verwendet werden.

5.1.2.1. Vorteile von NodeJs

Einer der größten Vorteile von *NodeJS* gegenüber anderen Web-Serving-Techniken ist, dass es sich bei der Kommunikation um eine **Non-Blocking I/O** also Einer **asynchronen** Kommunikation handelt.

5. Backend

Außerdem wird keine zusätzlicher Server benötigt bei einer NodeJS Anwendung, da die Anwendung auch den Webserver darstellt. Somit haben Server und Webserver die selbe Programmiersprache *Javascript* was wiederum ein großer Vorteil ist.

NodeJs beinhaltet außerdem eine Menge **Build-in-Modulen**, die ohne zusätzliche Installation zur Verfügung stehen. Zwecks Installation, Entfernung und Aktualisierung von zusätzlichen Packages verfügt Node über den sogenannten *Node Package Manager (npm)*. Alternativ kann auch der Packetmanager *Yarn* verwendet werden.

Packages sind externe Bibliotheken die untergeordnete Aufgaben in Projekten durchführen.

5.1.3. NodeJs Anwendungsbereich

Umfragen des Codings-Forums *Stack Overflow* ergab das rund 50.4 der professionellen Entwickler NodeJs als Backend Framework verwenden.

Große Unternehmen wie:

- Netflix
- Paypal
- Uber
- Linkedin

verwenden ebenfalls das Framework. [26] [27]

5.2. Datenbank

5.2.1. Definition

Eine Datenbank ist eine große Sammlung von strukturierten und organisierten Daten, die in einem Computersystem gespeichert sind. Gesteuert wird eine solche Datenbank von einem **Datenbankmanagementsystem (DBMS)**. Die Daten und das DBMS werden zusammen als Datenbanksystem bezeichnet, jedoch meistens als Datenbank abgekürzt.

Die Daten in den gängigsten verwendeten Datenbanken werden in Zeilen und Spalten unterteilt und in einer Tabelle modelliert. Aufgrund dieser Struktur können diese Daten dann einfach, abgerufen, modifiziert, aktualisiert, kontrolliert und organisiert werden. Fast alle Datenbanken verwenden zum Abfragen oder Schreiben von der *Structured Query Language (SQL)*. [28]

5.2.2. Datenbankmanagementsystem

Als *Datenbankmanagementsystem(DBMS)* wird eine Datenbankprogramm bezeichnet die als Schnittstelle zwischen Usern und der Datenbank dient. Es ermöglicht den Benutzern Informationen aufzurufen zu aktualisieren oder zu optimieren. Administrative Aufgaben werden durch so ein *DBMS* ebenso erleichtert. Abläufe wie Leistungsüberwachung, Abstimmung sowie Backup und Wiederherstellung werden ermöglicht.

Die wohl bekanntesten *DBMSs* sind:

5. Backend

- MySQL
- Microsoft Access
- Microsoft SQL Server
- File Maker Pro
- Oracle database
- dBase

5.2.3. Structured Query Language (SQL)

Die Programmiersprache SQL wird heutzutage von fast allen relationalen Datenbanken verwendet um Daten Abzufragen, Manipulieren und zu Definieren. SQL hat außerdem zu vielen Erweiterungen bei großen Unternehmen wie IBM, Oracle und Microsoft.

5.2.4. MySQL

MySQL ist ein relationales Datenbankmanagementsystem, das auf dem Open-Source Programmiersprache *SQL* basiert. Eingeführt wurde es 1995 vom schwedischen Unternehmen *MySQLAB* und wurde dann im Jahre 2010 von *Oracle* übernommen. Der Name fügt sich aus der Kombination "My", der Name einer Tochter eines Mitbegründers und "SQL" der Abkürzung für Structured Query Language zusammen.

Um das Prinzip von *MySQL* zu verstehen muss man zwei Konzepte verstehen.

5.2.4.1. Relationale Datenbanken

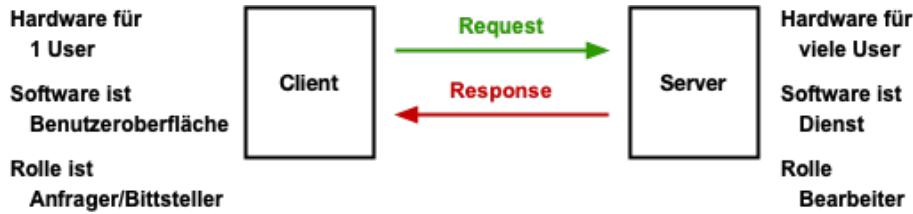
Bei Relationalen Datenbanken werden die Daten in Form von Tabellen dargestellt, anstelle von einem großen Speicherbereich. Bei relationalen Datenbanken gibt es einen sogenannten **Schlüssel**. Mithilfe dieses Schlüssels ist es möglich, Daten zwischen verschiedenen Tabellen miteinander zu verknüpfen. In der Regel ist ein solcher Schlüssel eine **eindeutige Identifikationsnummer (ID)**.

5.2.4.2. Client-Server-Model

MySQL basiert außerdem auf dem Prinzip des **Client-Server-Model**. Im **Server** befinden sich die Daten und um auf diese zugreifen zu können wird ein sogenannter **Client** benötigt. Unter *SQL* sendet also der Client eine Anfrage auch genannt **request** worauf der Datenbankserver mit den benötigten Daten antwortet **response**.

Wir haben uns für eine *MySQL* Datenbank entschieden, da wir uns schon in der Schule damit auseinander gesetzt haben und wir somit schon Erfahrung haben.

5. Backend



5.2.5. Andere Datenbanktypen

Wie bereits oben beschrieben ist *MySql* ein relatives Datenbanksystem. Es gibt jedoch einige weiter Datenbanktypen.

5.2.5.1. Objektorientierte Datenbanken

Die Informationen werden in einer objektorientierten Form von Objekten dargestellt.

5.2.5.2. Verteilte Datenbanken

In einer verteilten Datenbank werden die Informationen in zwei oder mehreren Dateien gespeichert. Diese Dateien können auf mehreren Rechnern verteilt sein, die entweder an einem einzigen Standort sind oder welche über ein Netzwerk miteinander kommunizieren.

5.2.6. Data Warehouse

Data Warehouse Datenbanken bestehen aus einem zentralen Daten Repository. Es wurde speziell für schnelle Anfragen und Analysen konzipiert.

5.2.7. NoSQL-Datenbanken

Im Gegensatz zu einem relativen Datenbanksystem ermöglicht eine *NoSql* Datenbank, semistrukturierter Daten bzw. unstrukturierte Daten zu speichern und verändern. *NoSql* Datenbanken setzen zur Organisation nicht auf sogenannte **keys**, sondern auf Wertepaare, Objekte, Dokumente, Listen oder Reihen. Da *NoSql* Systeme sehr flexibel sind, eignen sie sich besonders für große Datenmengen sogenannte **Big Data**. Die bekanntesten *NoSql* Datenbanksysteme sind

- Apache
- Cassandra
- Riak
- MongoDB
- Redis
- CouchDB

5. Backend

5.2.8. Diagrammdatenbanken

Daten werden anhand von Entitäten und ihren Beziehungen abgespeichert.

5.2.9. OLTP-Datenbanken

Eine *OLTP* Datenbank kennzeichnet sich dadurch, dass sie schnell analytisch und für eine Vielzahl von Transaktionen mehrerer Benutzer ausgelegt ist.

Das sind natürlich nicht alle vorhandenen Datenbanktypen. Die Anderen sind jedoch nicht weit verbreitet und speziell für wissenschaftliche, finanzielle oder andere Funktionen konzipiert.

5.2.9.1. Password-Hashing

Damit in der Datenbank die Passwörter nicht im Klartext angezeigt werden. Müssen sie vorher **gehashed** werden.

Was bedeutet Hashing? Das sogenannte Password-Hashing dient einem erhöhten Datenschutz. Er wandelt das Klartext Passwort in eine Reihe von Zeichen und Symbolen-folgen an.

Wie funktioniert das Hashing? Mittels eines Password Hashing Verfahrens werden Passwörter in einer festgelegten Codefolge in zufälligen Buchstaben und Zahlen umgewandelt. Die Generierung der sogenannten *Hashes* läuft also automatisiert mit einem Hash-Algorithmus ab. Die bekanntesten Hashing Algorithmen sind **Skrypt** oder **Argon2**.

Salted Passwords Da ein Algorithmus gleiche Passwörter immer gleich Hashed und man somit leicht das Klartext Passwort raus filtern kann wird der Hash noch "gesalzen". Ein *Salt* ist eine zufällig generierte Ziffer die in dem Hash einfließt. Somit kann also verhindert werden das der gleiche Hash öfter in der Datenbank steht. [29]

5. Backend

```
1   router.post('/register', async (req, res, next) => {
2     const password = req.body.password;
3     const encryptedPassword = await bcrypt.hash(password,
4       saltRounds);
5     try {
6       const temp = new User(
7         null,
8         req.body.name,
9         encryptedPassword,
10        req.body.email,
11      );
12      var alreadyExists = await User.alreadyExists(
13        con,
14        temp.name,
15        temp.email,
16      );
17      if (!alreadyExists) {
18        await User.insertValue(con, temp);
19        const user = await User.getByEmail(con, temp.
20          email);
21        token = User.generateToken(user);
22        res.send({ success: true, token: token });
23      } else {
24        res.send({ success: false, message: 'User
25          already exists!' });
26      }
27    } catch (e) {
28      console.log(e);
29      res.sendStatus(500);
30    }
31  });
32});
```

Code-Snippet 5.1.: Code-Snippet-Hashing

Erklärung zum Code: In Zeile zwei wird das Klartext Passwort aus dem Body entnommen. Des weiteren wird in der nächsten Zeile das Klartext Passwort mittels der asynchronen Methode `bcrypt.hash(klartext Passwort, saltRounds)`. Die saltRounds sind ein Faktor der bestimmt wie lange es dauert um ein einziges BCrypt Hash zu erstellen. Den Faktor um 1 zu erhöhen würde eine doppelt so lange Hash Zeit bedeuten. Im Weiterfolgenden Code wird zuerst gecheckt ob ein User mit der Email oder dem Username bereits besteht. Falls der Username und die Email unique sind wird der Benutzer in die Datenbank eingetragen und ein Token 14.2 wird erstellt. [30]

5.3. Docker

5.3.1. Was ist Docker?

Der Begriff *Docker* bezieht sich auf mehrere Dinge. Ein Docker ist sowohl ein Open Source Community Projekt, als auch das Unternehmen *Docker Inc*, dass das Projekt und seine Tools hauptsächlich unterstützt.

Unter der IT-Software Docker versteht man also eine Containertechnologie, die die Erstellung von Linux Containern ermöglicht. Die Container sind in der Regel leichtgewichtige, modulare virtuelle Maschinen. Dadurch entsteht vor allem Flexibilität, da die Container einfach erstellt, eingesetzt, konzipiert und zwischen Umgebungen bewegt werden können.

5.3.2. Wie genau funktioniert Docker?

Damit Docker Prozesse unabhängig voneinander ausführen kann, verwendet die Docker Technologie den Linux Kernel und seine Funktionen wie **Cgroups namespaces**. Somit werden die einzelnen Container voneinander isoliert und es ist möglich mehrere Prozesse und Apps getrennt voneinander ausführen zu können. Dadurch entsteht eine bessere Infrastruktur die gleichzeitig Sicherheit bewahrt.

Die Containertools arbeiten mit einem Image-basierten Bereitstellungsmodell. Aufgrund dessen ist es einfach Anwendungen oder Pakete mit all ihren Abhängigkeiten in mehreren Systemen gemeinsam zu benutzen. Alle diese Tools bauen auf Linux Container auf was ihre Benutzung einfach und einzigartig macht. Außerdem ermöglicht es eine schnellere Bereitstellung und Kontrolle von Versionen und ihrer Verbreitung.

5.3.3. Unterschied Docker Container und herkömmliche Linux-Container

Ursprünglich basierte die Docker Technologie auf der **LXC- Technologie**, die mit den traditionellen Linux Container assoziiert wird. Docker hat sich jedoch mittlerweile von dieser Abhängigkeit befreit, da LXC keine gute Entwickler und Nutzererfahrung bot. Docker bietet wie bereits erwähnt mehr Funktionen an als nur die reine Ausführung von Container, sie vereinfacht den Prozess der Erstellung und des Aufbaus von Container sowie den Versand von Images.

Linux Container verwenden ein init-System, das mehrere Prozesse verwalten kann. Somit können komplett Anwendungen als eines betrieben werden. Docker hingegen schlüsselt die Anwendungen und ihre einzelnen Prozesse auf und stellt somit die Tools bereit.

5.3.4. Vorteile von Docker

Ein großer Vorteil von Docker ist, dass bei einer Reparatur oder Aktualisierung nur ein Teil der Anwendung außer Betrieb genommen werden muss und der Rest weiter laufen kann. Außerdem ermöglicht es Docker Prozesse in mehreren Apps gemeinsam zu verwenden.

5. Backend

5.3.4.1. Layer und Image

Ein Docker-Image besteht aus mehreren Layer. Mehrere Layer werden dann in einem einzelnen Image vereint. Falls ein User einen Befehl wie *copy* oder *run*, wird ein neuer Layer erstellt.

Diese Layer verwendet Docker dann für die Erstellung neuer Container, was eine enorm schnelle Entwicklung bedeutet. Bei jedem Image hat man außerdem ein eingebautes Änderungsprotokoll und somit volle Kontrolle über Container-Images.

5.3.4.2. Rollback Funktion?

Eines der größten Vorteile vom Layering Prinzip ist, das man auf vorherige Versionen Zurücksetzen kann. Aufgrund diesen Ansatz wird für eine kontinuierliche Integration und Bereitstellung gesorgt. **Continuous Integration/Continuous Development CI/CD**

5.3.4.3. Schnelle Bereitstellung

Die neue Bereitstellung von Hardware dauert in der Regel Tage. Mithilfe der Docker-basierten Container wird die Bereitstellung in Sekunden erledigt. Da für jeden Prozess ein neuer Container erstellt wird kann man einfach ähnliche Prozesse in Sekunden teilen. Aufgrund das beim Hinzufügen oder Verschieben eines Containers das Betriebssystem nicht neu gebootet werden muss sind die Bereitstellungszeiten wesentlich kürzer.

[31]

Teil III.

Webseite

6. Allgemeines

6.1. Idee

Die Idee hinter der Webseite war es, einen schnellen Überblick zu erschaffen. Skater sollen schnell Skateparks finden können und auch ihre Meinung zu den jeweiligen Parks teilen können. Außerdem sollten sie in der Lage sein einen Vorschlag von einem Park an die Seitenbetreiber zu schicken, welchen diesen dann in die Datenbank hinzufügen können. Die Administration unserer Anwendung erfolgt ebenfalls über die Seite.

6. Allgemeines

6.2. Daten bekommen

Auf der Website werden wiederholt Daten vom Server geholt. Dies mache ich mit einem *fetch*. Mit dem Fetch Befehl und den richtigen API-Link konnte ich nur so immer die Daten holen, die ich gebraucht habe.

Im Projekt selbst habe ich einen Custom Fetch verwendet, welcher zusätzlich zu den Daten auch einen Error liefert, falls der Fetch fehlschlägt, als so wohl auch eine boolean Variable, welche true ist und erst false wird, wenn alle Daten vom Server an der Website angekommen ist. Diese Variable genannt *isPending* ist in dem Sinne von nützen, dass der Nutzer informiert wird, dass die Daten noch laden müssen.

```
1 import { useState, useEffect } from 'react';
2 import Error from '../staticViews/Error'
3
4 const useFetch = url => {
5   const [data, setData] = useState(null);
6   const [isPending, setIsPending] = useState(true);
7   const [error, setError] = useState(null);
8
9   useEffect(() => {
10     const abortCont = new AbortController();
11
12     setTimeout(() => {
13       fetch(url, {
14         method: 'GET',
15         signal: abortCont.signal })
16       .then(res => {
17         if (!res.ok) {
18           throw Error('Daten konnten nicht empfangen werden');
19         }
20         return res.json();
21       })
22       .then(data => {
23         setIsPending(false);
24         setData(data);
25         setError(null);
26       })
27       .catch(err => {
28         if (err.name === 'AbortError') {
29           console.log('fetch abgebrochen')
30         } else {
31           setIsPending(false);
32           setError(<Error></Error>);
33         }
34       });
35     });
36
37     return () => abortCont.abort();
38   }, [url]);
39
40   return { data, isPending, error };
41 };
42
43 export default useFetch;
```

Wie man im Code sehen kann, werden die drei Variablen mit einem *useState* und dem Wert null

6. Allgemeines

deklariert. Außer *isPending* welche wie oben schon genannt standardmäßig auf true ist. Diese drei Werte werden am Ende als return Wert zurückgegeben um sie dann außerhalb des useFetch benutzen zu können.

Falls ein Fehler auftritt, wird der State der Error Variable auf eine Error Komponente gesetzt. Dies habe ich gemacht um bei jedem Fetch dem Benutzer eine Einheitliche Fehlermeldung zu geben, falls ein Fehler auftritt.

Diese Fehlermeldung sieht wie folgt aus:

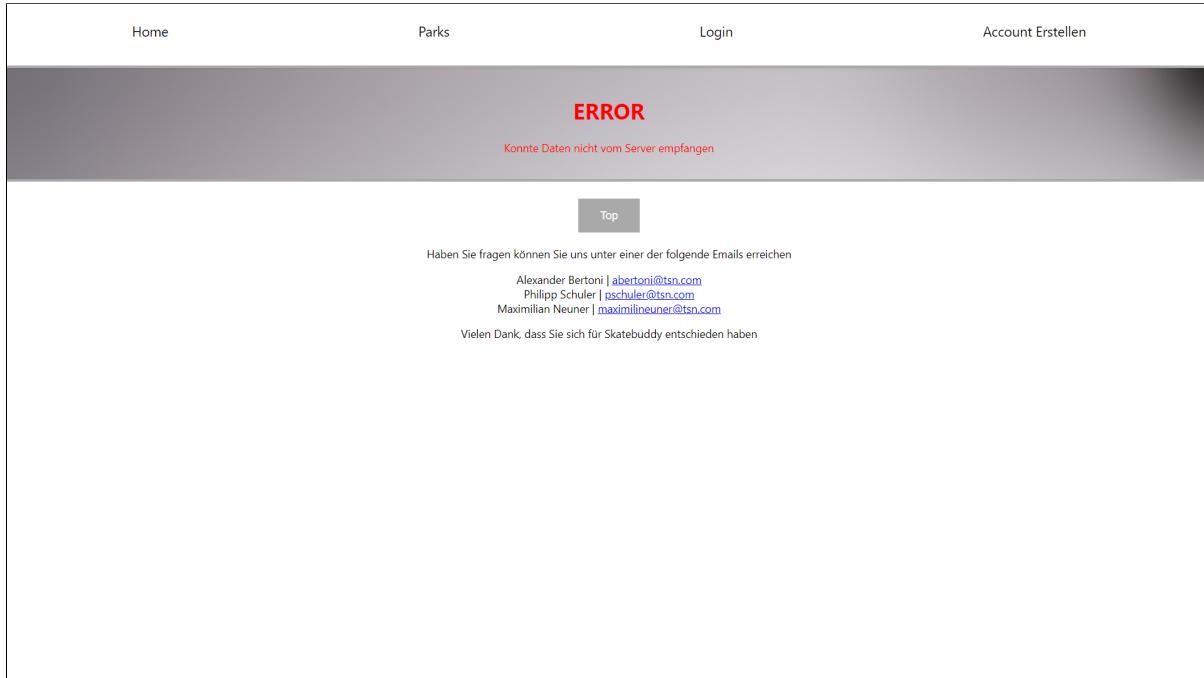


Abbildung 6.1.: Fehlermeldung wenn das Daten holen vom Server fehlschlägt

6.3. Daten senden

Natürlich kann man über die Webseite auch Daten an den Server senden. Dies wird benötigt beim Erstellen eines Benutzers sowohl als auch beim einloggen und beim Erstellen eines Parks oder einer Review zu einem Park. Das Senden der Daten an den Server erfolgt ebenfalls mit einem *Fetch* diesmal jedoch ist der fetch keine *get Method* wie bei dem useFetch, sondern eine *post Method*.

```
1  fetch("APU_URL", {
2    method: 'POST',
3    headers: {"Content-Type": "application/json"},
4    body: JSON.stringify(data)
5  },
6  ).then(function(response) {
7    return response.json();
8  })
```

6. Allgemeines

Im header geben wir an, welchen Datentyp (In unserem Fall ein JSON) wir senden. Im body wandeln wir unseren Datenstring in ein JSON um und senden diesen dann an die entsprechende URL. Dann warten wir auf eine Antwort vom Server, und returnen diese als json. Diese response können wir dann nach beliebigen verwenden.

6.4. Benutzertoken

Der Benutzer kann sich mittels eines einfachen Formular in die Webseite einloggen als auch einen neuen Account erstellen. Bei erfolgreichem Login oder Account erstellen erhält man vom Server einen *JWT (JSON Web Token)*, welcher für die weitere Verwendung dekodiert werden muss. Dieser kodierte Token wird im LocalStorage gespeichert, damit man sich nicht jedes mal neu einloggen muss, wenn man die Seite verlässt.

```
1 localStorage.setItem("user", JSON.stringify(response.data));
```

So wurde der Token in den LocalStorage der Seite gespeichert.

TODO: Name Ref fauler neuner zu Token

6.4.1. Token dekodieren

Das dekodieren von dem Token erfolgt in einer *decodeToken* Methode. In dieser Methode wird der Token mittels *post fetch* an den Server gesendet. War das Decoden des Tokens erfolgreich, erhalten wir als Antwort vom Server alle Daten zu dem eingeloggten User. Diese Daten werden dann in dem SessionStorage des Browsers gespeichert und werden somit beim schließen des Browsers wieder verworfen. Das dekodieren eines Tokens wird nach dem Login und bei dem Aufruf der Seite gemacht.

```
1 sessionStorage.setItem("data", JSON.stringify(data))
```

So erfolgt das Speichern der Tokendaten in den SessionStorage.

6.4.2. Token Überprüfung

Die Token selbst besitzen ein Ablaufdatum. Ist dieses Ablaufdatum erreicht, sind sie nicht mehr gültig und man wird ausgeloggt und muss sich wieder neu einloggen. Die Überprüfung dieses Ablaufdatum findet jedes mal statt, wenn die Seite neu aufgerufen wird. Wie auch schon beim Decoden eines Tokens, ist dies auch ein *post Fetch*, welches den Token an Server sendet. Ist das Ablaufdatum des Token erreicht, gibt der Server als Antwort *success:false* zurück. Ist dies der Fall ist der Token abgelaufen und man wird ausgeloggt. Ist der Token nicht abgelaufen gibt der Server die Antwort *success:true* und bleibt eingeloggt.

6.4.3. Token entfernen

Beim Ausloggen wird der Token aus dem LocalStorage entfernt. Die Information des dekodierten Token wird ebenfalls aus dem SessionStorage entfernt. Dies erfolgt ganz einfach mittels einer *logout* Methode, welche wie folgt aussieht:

6. Allgemeines

```
1 logout() {
2     localStorage.removeItem("user");
3     sessionStorage.removeItem("data")
4     sessionStorage.removeItem("validate")
5 }
```

Damit befindet sich nichts mehr im Local- und SessionStorage.

6.5. Index.js

Die *index.js* File ist eine von React autogenerierte Datei. In der Index Datei wird die App.js-Komponente mittels Renderfunktion gerendert. Sie wird innerhalb einer *React.StrictMode* Komponente geladen, welche dem Entwickler potenzielle Fehler anzeigt. Die Index Datei verfügt auch über eine vorgenerierte CSS Datei, welche verschiedenen Schriftarten enthält.

6.6. App.js

In der App Komponente ist die Navigationsleiste und der Fußzeile definiert, also sowohl auch die URLs zu den einzigen Views. Diese URLs sind mittels einen React Router definiert. Da der Footer direkt in der App-Komponente ausprogrammiert ist, befindet sich die CSS auch in der CSS-Datei der App-Komponente. Die Navigationsleiste wird genau wie die Views als Komponente reingeladen.

6.6.1. React Router, Routes, Route

Alle Komponenten aus dem die Webseite besteht sind innerhalb einer Router Komponente definiert. Die Router Komponente sorgt dafür, dass die Seite wie eine Single-Page-App funktioniert. Betätigt der Benutzer einen Wechsel der View, was normalerweise einen Seitenwechsel verursachen würde, wird die Anfrage an den Server vom React Router abgefangen und der Router sendet die neue View ohne einen Seitenwechsel zu verursachen.

Innerhalb der Router-Komponente, nämlich dort wo die View angezeigt werden soll (in meinem Fall unter der Navigationsleiste) ist die Routes Komponente definiert.

```
1 <Routes>
2     <Route path="/" element={<Home />} />
3     <Route path="/parks" element={<Parks />} />
4     <Route path="/skateparks/:id" element={<ParkDetails />} />
5     <Route path="/LogIn" element={<LogIn />}/>
6     <Route path="/CreateAccount" element={<CreateAccount/>}/>
7     <Route path="/AllMap" element={<AllMap/>}/>
8     <Route path="/Profile" element={<Profile/>}/>
9     <Route path="/AddPark" element={<AddPark/>}/>
10    <Route path="/AddRecommendation" element={<AddRecommendation/>}/>
11    <Route path="/Recommendations" element={<Recommendationlist/>}/>
12 </Routes>
```

Innerhalb der Routes Komponente werden die einzelnen Routen definiert. Je nachdem welche URL aufgerufen wird, wird eine andere Komponente geladen. Diese Routen werden mittels

6. Allgemeines

Link-Komponenten in den einzelnen Views geändert. Zum Beispiel wird bei der URL `/parks` die Komponente **Parks** als Inhalt geladen. Wie man sieht ist bei der `/skateparks` URL ein `/:id` am Ende. Dieses `:id` bedeutet, dass je nach welcher id man in die URL schreibt also 1, 2, 3, ... man der ParkDetails-Komponente einen anderen Parameter übergibt.

6.6.1.1. Link-Komponenten

Die Link-Komponente dient dazu die URL zu ändern falls man einen Viewwechsel durchführen möchte.

```
1 <Link to="/" className='home-nav' id="nav">
2     Home
3 </Link>
```

Durch das Klicken auf das Wort *Home* wird die URL in diesem Beispiel auf `/` geändert. Mit dieser geänderten URL ändern der Router mit den definierten Routes die View.

6.6.2. Navigationsleiste

Die Navigationsleiste ist genauso wie der Fußzeile immer sichtbar. Sie steht über dem Inhalt der Seite und dient zur Navigation der Webseite. Die Navigation erfolgt ganz einfach über Link-Komponenten.



Abbildung 6.2.: Navbar der Webseite wenn man nicht eingeloggt ist

Das Design der Navigationsleiste ist wie auch der Rest der Seite sehr simpel gehalten. Schwebt man mit der Maus über eins der Felder mit denen man die View wechseln kann, wird dieses unterstrichen.

Klickt man auf Home, Parks, Login oder Account Erstellen wird die Link-Komponente welche dich zu den jeweiligen Views bringt aufgerufen. Ist man auf der Webseite jedoch eingeloggt, verschwinden jedoch die Login und Account erstellen Felder und es erscheint ein Profil Feld.



Abbildung 6.3.: Navbar der Webseite wenn man eingeloggt ist

6. Allgemeines

Das Überprüfen ob man eingeloggt ist oder nicht erfolgt ganz einfach über JSX.

```
1  {token &&
2      <Link to="/Profile" className='pofile-nav' id="nav">
3          Profile
4      </Link>
5  }
6
7  {!token &&
8      <>
9          <Link to="/LogIn" className='login-nav' id="nav">
10             Login
11         </Link>
12         <Link to="/CreateAccount" className='createAccount-nav' id="nav">
13             Account Erstellen
14         </Link>
15     </>
16 }
```

Ist die token Variable nicht *null* bzw nicht *false*, wird der Code in den geschwungenen Klammern mit token ausgeführt. Ist die Variable *null oder false* wird der Code mit dem !token ausgeführt. Die Variable token beinhaltet dies, was in dem SessionStorage der Seite steht. Ist man also eingeloggt, ist diese nicht *null* und der Code mit dem Profile-Link wird ausgeführt. Ist man nicht eingeloggt ist token *null* und der Code in den Klammern mit !token wird ausgeführt.

6.6.3. Fußzeile

Die Fußzeile ist ein simpler Absatz mit Information am Ende des Inhaltes. Sie ist wie auch die Navigationsleiste auf jeder View die selbe. In der Fußzeile befindet sich neben der Information auch ein Top-Knopf, welcher bei Betätigung ganz nach oben scrollt.

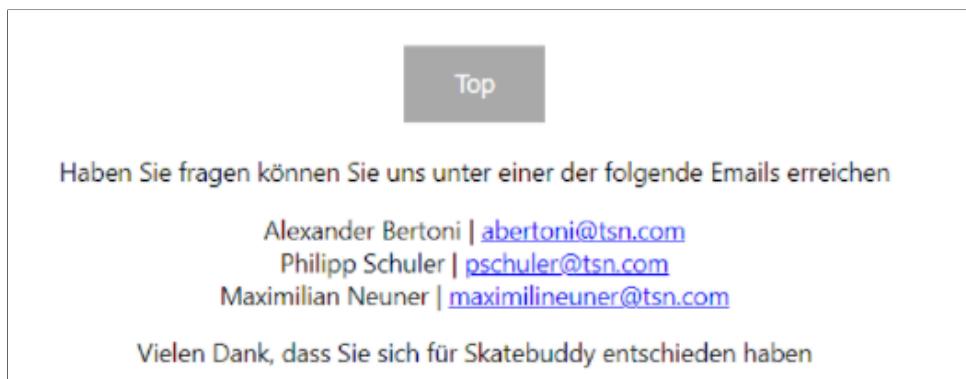


Abbildung 6.4.: Die Fußzeile der Webseite

6. Allgemeines

Damit der Knopf bei Mausklick von selbst nach oben scrollt, weiß ich ihm eine JavaScript-Funktion bei Knopfdruck zu.

```
1 function topFunction() {
2     document.body.scrollTop = 0;
3     document.documentElement.scrollTop = 0;
4 }
5
6 <button onClick={topFunction} id="myBtn" title="Go to top" className="top-
button">Top</button>
```

Bei Knopfdruck wird diese simple Funktion ausgeführt, welche ganz nach oben scrollen. Damit die Funktion scrollt und nicht auf einmal ganz oben ist, musste noch folgendes in der CSS Datei hinzugefügt werden.

```
1 html {
2     scroll-behavior: smooth;
3 }
```

Damit springt die Seite mit Knopfdruck nicht nach oben, sondern scrollt reibungslos.

6.7. Slideshow

Auf der Startseite und auf der Liste der Parks gibt es eine Slideshow von Bildern. Diese Slideshow Komponente ist extern und habe ich mittels npm heruntergeladen und in meinem Projekt importiert. Der Ersteller dieser Komponente ist *femioladeji*. Verwenden tu ich diese Komponente wie folgt:

```
1 const Slideshow = (parkpics) => {
2     return (
3         <div className="slide-container">
4             <Slide autoplay={false} transitionDuration={420}>
5                 {parkpics.parkpics.map(pic=> (
6                     <div className="each-slide" key={pic.skateparkId}>
7                         <div style={{'backgroundImage': `url(${https://skate-buddy.josholaus.
8                             com/api/skateparkpictures/${pic.skateparkId}/${pic.picId}})`}}
9                             className='bg'>
10                         </div>
11                     </div>
12                 )));
13             </Slide>
14         </div>
15     )
16 }
```

Wie man sieht übergebe ich der JavaScript Funktion eine Parameter names *parkpics*. Die Daten für diesen Parameter hole ich mir ganz einfach mittel *useFetch*. In diesem Parameter befindet sich ein zwei-dimensionales Array, welches die ParkID des Parks also sowohl auch die ID des Bildes enthält. Insgesamt übergib ich der der Slide Komponente zwei Parameter: *autoplay=false* sorgt dafür, dass die Bilder nicht von selbst weiter schalten. (Auf der Startseite ist diese Variable auf true) und eine *transitionDuration* welche die Geschwindigkeit festlegt, mit der die Bilder weiter geschaltet werden soll. Innerhalb der Slide Komponente steht eine map Funktion,

6. Allgemeines

welche für jedes Element im Parameter parkpics ein div erzeugt, welche das jeweilige Bild als Hintergrundbild besitzt.

6.8. Map

Auf der Webseite habe ich die Google-Maps-API benutzt, um eine schnelle Überblick von allen Parks zu schaffen. Die Google Maps Komponente wurde wie die Slideshow Komponente über npm herunter geladen. Der Name der Komponente lautet *google-map-react* und wurde von *istarkov* und *itsmichaeldiego* erstellt. Ich habe auf der Seite zwei mal die Map benutzt. Der folgende Code ist die Map, wo alle Parks eingezeichnet sind.

```
1 const options = (maps) => {
2     return {
3         minZoom: 9,
4         maxZoom: 20,
5         disableDefaultUI: true,
6         mapTypeControl: true,
7         streetViewControl: true,
8         styles: [{ featureType: 'poi', elementType: 'labels', stylers: [
9             { visibility: 'on' } ] }],
10    };
11 }
12 const defaultProps = {
13     center: {
14         lat: 47.2683,
15         lng: 11.3933,
16     },
17     zoom: 11,
18 };
19
20 <GoogleMapReact
21     options={options}
22     bootstrapURLKeys={{ key: Keys }} //API-Key
23     defaultCenter={defaultProps.center}
24     defaultZoom={defaultProps.zoom}>
25     {parks.map(park => (
26         <Marker
27             lat={park.latitude}
28             lng={park.longitude}
29             name={park.name}
30             color="red"
31             link={park.skateparkId}
32         />
33     )));
34     {UserLangitude && <Marker
35         lat={UserLangitude}
36         lng={UserLongitude}
37         name="Ihre Position"
38         color = "blue"
39     />}
40 </GoogleMapReact>
```

In den options, welche ich als Parameter der Komponente übergebe, wird der maximale und minimale Zoom definiert, der möglich sein soll. Außerdem werden viele Variablen mit dem

6. Allgemeines

Wert `true` übergeben, welche verschiedene Google-Maps Eigenschaften wie zum Beispiel Street view aktivieren. Der Api-Key wird mittels `useFetch` aus einer Textdatei ausgelesen, damit man diesem nicht im Quellcode zu finden ist. Innerhalb der `park` Variable, welche auch für die `map` verwendet wird, befinden sich alle Informationen aller Parks. In der Marker Komponente, wird dann der Breiten- und Längengrad als sowohl auch der Parkname als Parameter übergeben. Es wird auch die Park ID als Parameter übergeben, um Innerhalb der Marker Komponente einen Link zu den Details des jeweiligen Parks zu ermöglichen. Drückt man also auf den Marker auf eines gewissen Parks, landet man auf dessen Details. Auf der Map wird ebenso ein Marker geladen, dem der Längengrad und Breitengrad des Benutzers übergeben wird, um eine bessere Orientierung der Umgebung zu ermöglichen.

6.8.1. Marker

Die Marker selbst sind eine einfache React-Komponente, welche auf die Position des angegeben Längen- und Breitengrades platziert werden.

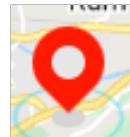


Abbildung 6.5.: Die Startseite

Den Namen den man den Markern als Parameter übergibt, wird angezeigt, wenn sich die Maus über den Marker befindet. Übergibt man den Marker eine ParkID als link, kann dieser dazu benutzt werden die View zu einer Park Details-View zu ändern. Übergibt man den Marker keinen Link Parameter passiert beim Drücken des Markers nichts.

6.9. Design

Für das Design der Webseite wurden überwiegend grau/weiße Boxen verwendet. Diese sehen wie folgt aus:

6. Allgemeines



Abbildung 6.6.: CSS Boxen

Diese Boxen haben einen einheitlichen CSS Code damit sie auf jeder View gleich aussehen.

```
1 display: block;
2 margin: 5% auto;
3 border-radius: 2vh;
4 box-shadow: inset 0 0 10em lightgray, 0 0 1em black;
5 background-color: white;
6 margin-top: auto;
7 margin-bottom: auto;
8 min-height: 70vh;
```

Die Box besteht also aus einem einfachen Border mit einem Schatten. Damit die Ecken schön rund sind, ist der Radius des Borders 2vh groß. Vh ist eine Einheit, welche sich auf die Höhe des Bildschirmes anpasst. 2vh bedeuten 2% der Bildschirmhöhe.

7. Views

7.1. Startseite

Wenn man die Webseite zum Ersten mal aufruft, landet man auf der Startseite (Home-view).

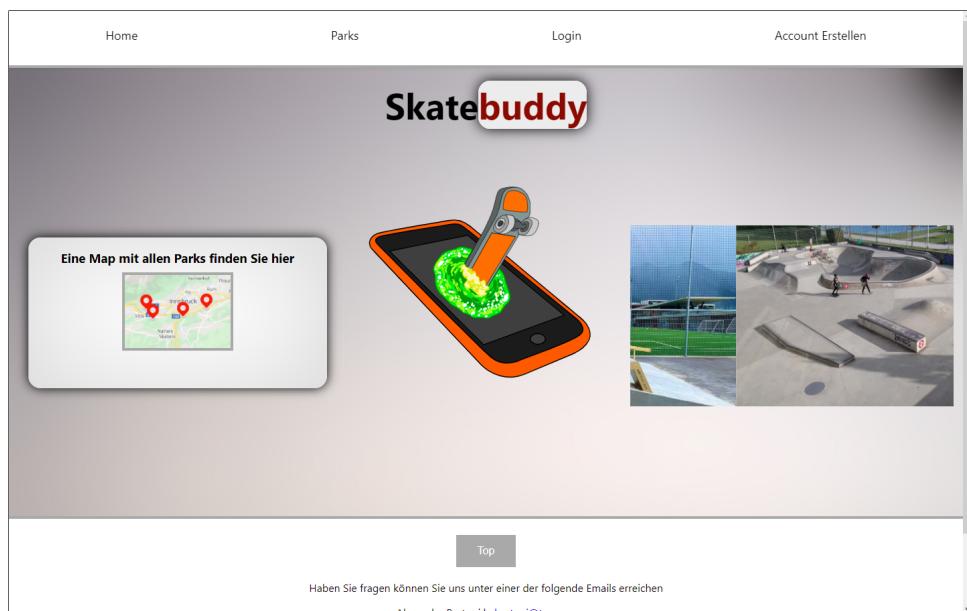


Abbildung 7.1.: Die Startseite

Hier findet man unser Projektlogo, als sowohl auch ein Knopf wo man zu einer Map kommt mit Überblick über alle Parks. Rechts vom Logo befindet sich eine Slideshow welche immer zwischen die ersten Bilder aller Parks umschaltet. Drückt man auf die Slideshow kommt man auf die Park Details Seite des Parks, dem das Bild gehört.

7.2. Account erstellen

Auf der Account erstellen Seite ist es möglich seinen eigenen Account zu erstellen. Hierzu wird ein einfaches HTML-Formular verwendet, welches bei *submit* die eingegebenen Daten an den Server sendet, als sowohl auch den Benutzer direkt einloggt. Existiert allerdings bereits ein User mit selben Namen beziehungsweise selbe Email, wird die Meldung *dieser Benutzer existiert bereits* ausgegeben. Um Tippfehler beim Passwort zu vermeiden, existiert ein Passwort bestätigen Feld. Die Überprüfung ob die Passwort bestätigen Zeichenkette die selbe ist wie die Passwort Zeichenkette erfolgt über die JavaScript Bibliothek **Yup** von *monastic.panic*.

7. Views

The screenshot shows a web page with a navigation bar at the top containing links for Home, Parks, Login, and Account Erstellen. Below the navigation is a large, semi-transparent gray overlay. In the center of this overlay is a white rectangular form titled "Erstellen Sie einen Account". The form has four input fields: "Name" (containing "Alexander"), "Email" (containing "abertoni@tsn.at"), "Passwort" (containing "*****"), and "Passwort bestätigen" (containing "*****"). Below the "Passwort bestätigen" field is a small red error message: "Passwörter stimmen nicht überein" (Passwords do not match). At the bottom of the form is a gray button labeled "Account erstellen". At the very bottom of the page, there is a small "Top" link and a footer message: "Haben Sie Fragen? Können Sie uns unter einer der folgenden Emails erreichen".

Abbildung 7.2.: Fehlermeldung für nicht Übereinstimmende Passwörter

7. Views

Um eine höhere Sicherheit zu garantieren, muss das Passwort mehr als sechs Zeichen beinhalten. Die Fehlermeldung für ein zu kurzes Passwort würde wie folgt aussehen.

The screenshot shows a web application interface. At the top, there is a navigation bar with links: Home, Parks, Login, and Account Erstellen. Below the navigation bar, there is a large, semi-transparent gray overlay covering most of the page. In the center of this overlay, there is a white rectangular form titled "Erstellen Sie einen Account". The form contains four input fields: "Name" (with the value "Alexander"), "Email" (with the value "abertoni@tsn.at"), "Passwort" (with the value "****"), and "Passwort bestätigen" (with the value "****"). Below the "Passwort" field, there is a red error message: "Muss mehr als sechs Zeichen beinhalten". At the bottom of the form is a gray button labeled "Account erstellen". In the bottom right corner of the main content area, there is a small "Top" link. At the very bottom of the page, there is a footer message: "Haben Sie Fragen? Können Sie uns unter einer der folgenden Emails erreichen".

Abbildung 7.3.: Fehlermeldung für nicht Übereinstimmende Passwörter

7. Views

7.3. Login

Existiert bereits ein Account ist es möglich sich über die Log In View mit diesem anzumelden. Dafür wird die Email als auch das Passwort benötigt. Hierzu wird wie auch beim Account erstellen eine simple HTML form verwendet.

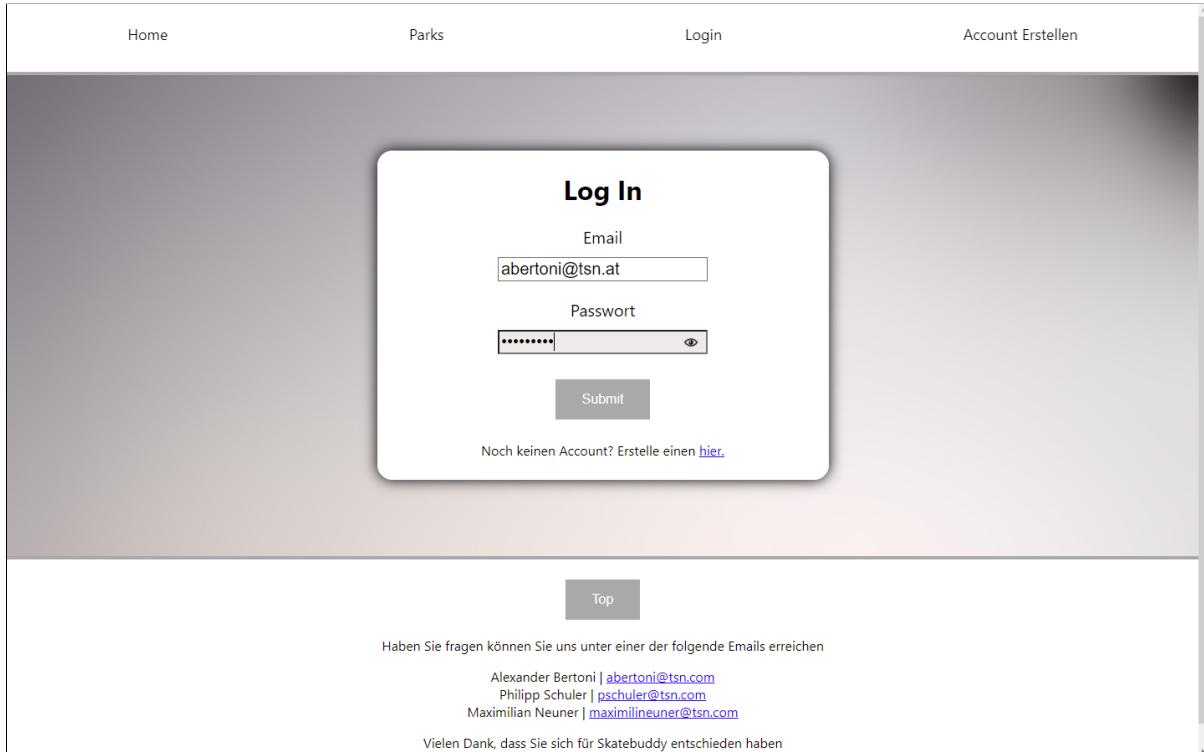


Abbildung 7.4.: Login Page mit verstecktem Passwort

Durch Mausklick auf das Auge in der Passwort Eingabe, lässt sich das Passwort für den Benutzer anzeigen. Damit lassen sich Tippfehler beim Eingeben erkennen, falls nötig.

7. Views

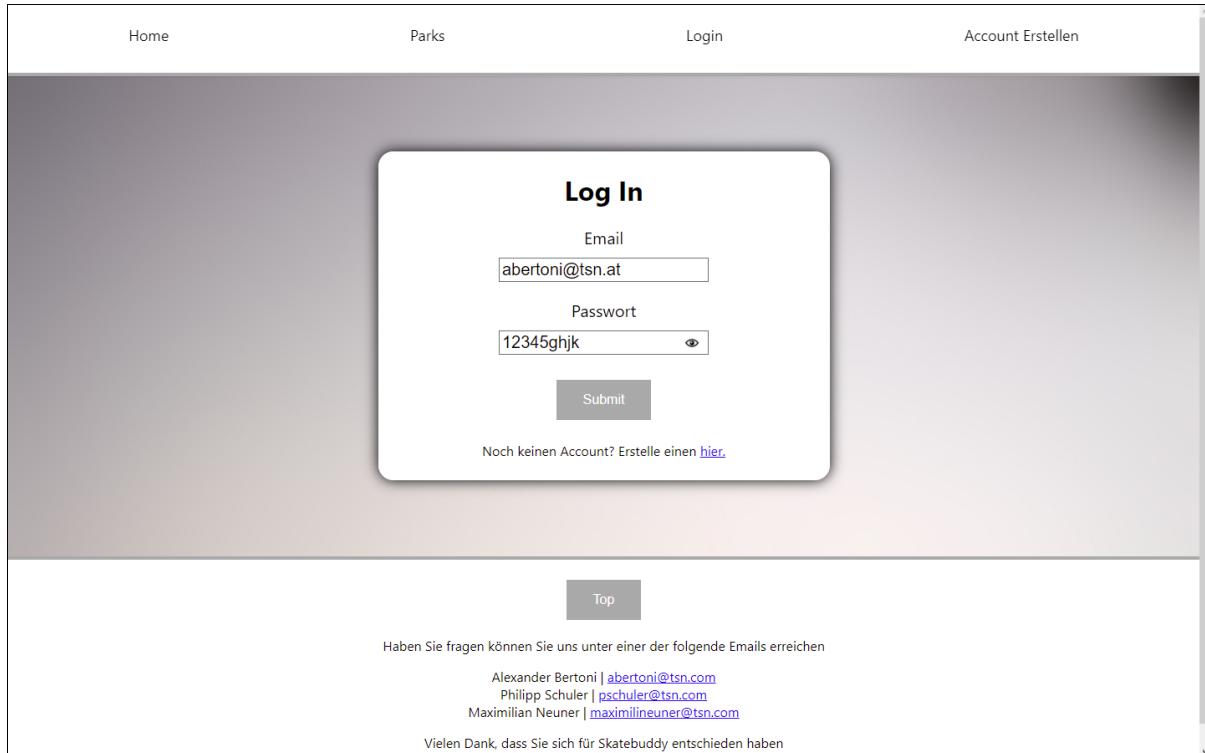


Abbildung 7.5.: Login Page mit gezeigtem Passwort

7.3.1. Augensymbol

```
1  .toggle-button {  
2      margin-top: 0;  
3      margin-left: -30px;  
4      height: 17px;  
5      width: 20px;  
6      border: white;  
7      background: no-repeat url('./eye.png');  
8      background-size: cover;  
9      cursor: pointer;  
10 }
```

```
1  const togglePassword = () =>{  
2      setPasswordShown(!passwordShown)  
3  }
```

```
1  <input className="input" type={passwordShown ? "text" : "password"} onChange={e  
= > setPassword(e.target.value)} />  
2  <button type="button" onClick={togglePassword} className="toggle-button"></br>  
   button>
```

7. Views

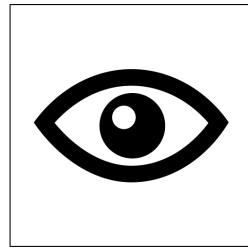


Abbildung 7.6.: Augensymbol zum Umschalten

Das Auge ist ein einfacher Knopf mit dem Bild als Hintergrund, welcher mit einer JavaScript Funktion eine Variable auf true setzt. Ist diese Variable auf *true* wird der Input-Type des inputs von *password* auf *text* umgeschaltet und das Passwort somit sichtbar.

7.4. Profil

A screenshot of a web application showing a user profile. At the top, there is a navigation bar with three items: "Home", "Parks", and "Profile". The "Profile" item is highlighted. Below the navigation bar, there is a large, semi-transparent gray overlay. In the center of this overlay is a white rounded rectangle containing a user's profile information. Inside this box, there is a small square thumbnail image of a person snowboarding. Below the thumbnail, the word "Name" is followed by the name "Alexander". Below that, the word "Email" is followed by the email address "abertoni@tsn.at". At the bottom of the profile box is a small gray rectangular button labeled "LogOut".

Abbildung 7.7.: Profil

Auf der Profilseite ist es möglich seinen Benutzernamen und seine Email einzusehen. Außerdem loggt man sich hier von der Seite wieder aus. Die Informationen über den Benutzer werden ganz einfach über seinen token ausgelesen. Ist man auf der Seite nicht eingeloggt und man versucht auf die Profilseite zu kommen indem man die URL dieser Seite eingibt, leitet diese auf die

7. Views

Login-view weiter. Die Überleitung auf die Login view erfolgt wie folgt:

```
1  useEffect(() => {
2      if(!sessionStorage.getItem("data")){
3          navigate("/LogIn");
4      }
5  );
```

UseEffect ist ein React Hook welcher ausgeführt wird, wenn die Seite einmal geladen wurde. Diese Überprüfung wird auch in der Admin-View der Seite verwendet.

7.5. Liste der Parks

Diese View gibt Auskunft über alle Parks, welche sich auf unseren Server befinden. Hier kann man den Namen, die durchschnittliche Bewertung und Bilder zu den Parks finden.

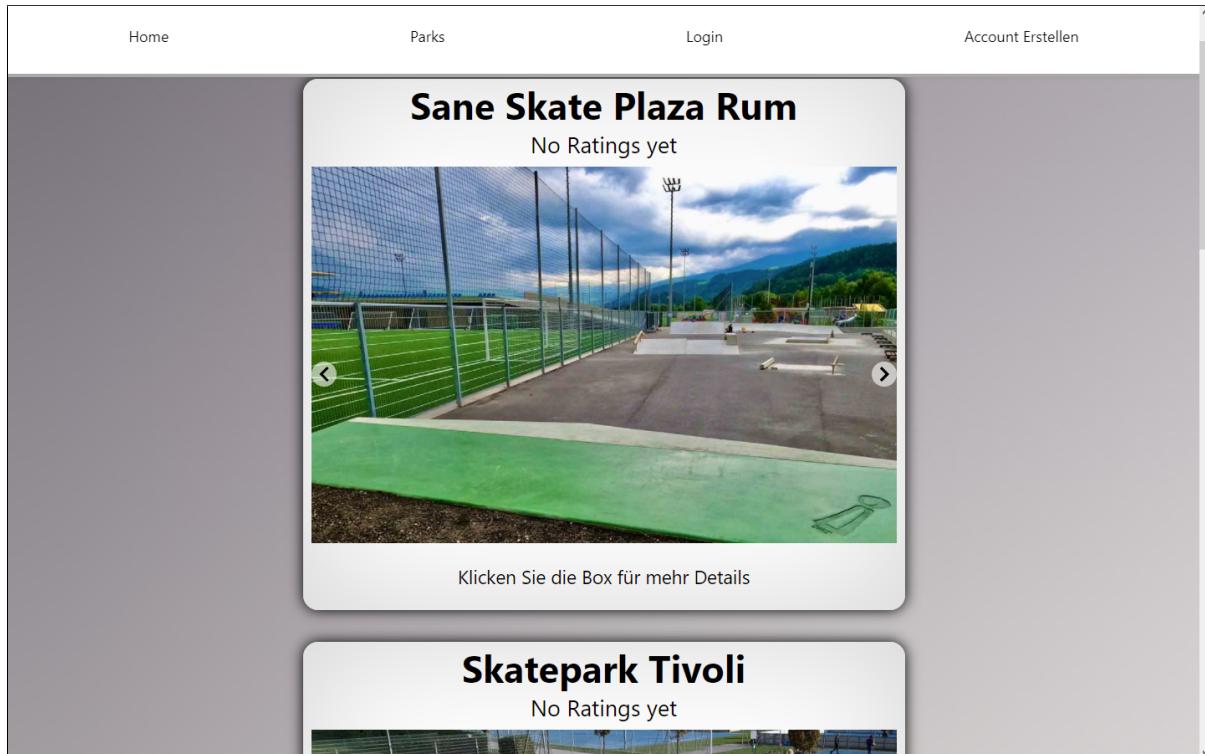


Abbildung 7.8.: Liste der Parks

Hier wurde die selbe Slideshow verwendet wie auf der Startseite. Drückt man auf einen der Boxen drauf, kommt man auf die Park Details des Parks, wo man die Bewertungen und Hindernisse, welche der Park besitzt einsehen kann.

7. Views

7.5.1. Suchleiste

Ganz oben auf der Parklisten Ansicht befindet sich eine Suchleiste, mit welche man nach einen bestimmten Park suchen kann.



Abbildung 7.9.: Liste der Parks

Wird in die Suchleiste eine Zeichenkette angegeben, werden nur noch die Parks angezeigt, welche diese Zeichenkette in ihrem Namen besitzen. Dabei wird Groß- und Kleinschreibung komplett ignoriert um ein leichteres Suchen zu ermöglichen. Dafür wird der Input ganz einfach bevor er mit den Namen verglichen wird auf Kleinbuchstaben umgeändert.

Ist ein Benutzer eingeloggt, befindet sich über der Suchleiste ein Knopf bei welchen man Vorschläge erstellen kann.

7.6. Vorschläge erstellen

A screenshot of a web form titled 'Erstelle einen Vorschlag'. The form is divided into several sections: 'Parkname' (input field), 'Adresse' (input field), 'Busstop' (input field), 'Koordinaten' (Latitude and Longitude input fields), 'Hindernisse' (dropdown menus for 'Gap' and '10', a 'Hinderniss hinzufügen' button, and a table). The table has columns 'Hinderniss', 'Schwierigkeit', and 'Entfernen'. It contains two rows: one for 'Quarter' with a value of '8' and one for 'Gap' with a value of '10'. Both rows have an 'Entfernen' button at the end. At the bottom is a 'Vorschlag speichern' button.

Abbildung 7.10.: Einen Vorschlag erstellen

7. Views

Einen Vorschlag erstellen kann jeder eingeloggte Benutzer. Diese Vorschläge können später von den Admins abgelehnt werden oder als Park hinzugefügt werden. Wie auch schon der Login oder das Account erstellen funktioniert auch das Vorschläge erstellen über ein HTML-Formular. Der Latitude muss sich jedoch innerhalb von +- 90 befinden und der Longitude innerhalb von +- 180. Die Hindernisse sind ein HTML-select welche bei Betätigung des Knopfes das Hindernis als sowohl auch die Schwierigkeit in eine Liste gespeichert. Diese Liste wird mittels map Funktion als HTML-Tabelle ausgegeben. Die Elemente dieser Liste erhalten eine temporäre Id. Diese temporäre Id wird beim *Entfernen* Knopf verwendet, da dieser das Element anhand der Id entfernt.

7.7. Admin-Page

Ist der Benutzer als Admin angemeldet erscheint unter dem Vorschlag erstellen Knopf ein *Vorschlagsverwaltung/Park hinzufügen*, welcher einen auf die *Recommendation-View* weiterleitet.

7.7.1. Recommendation-View

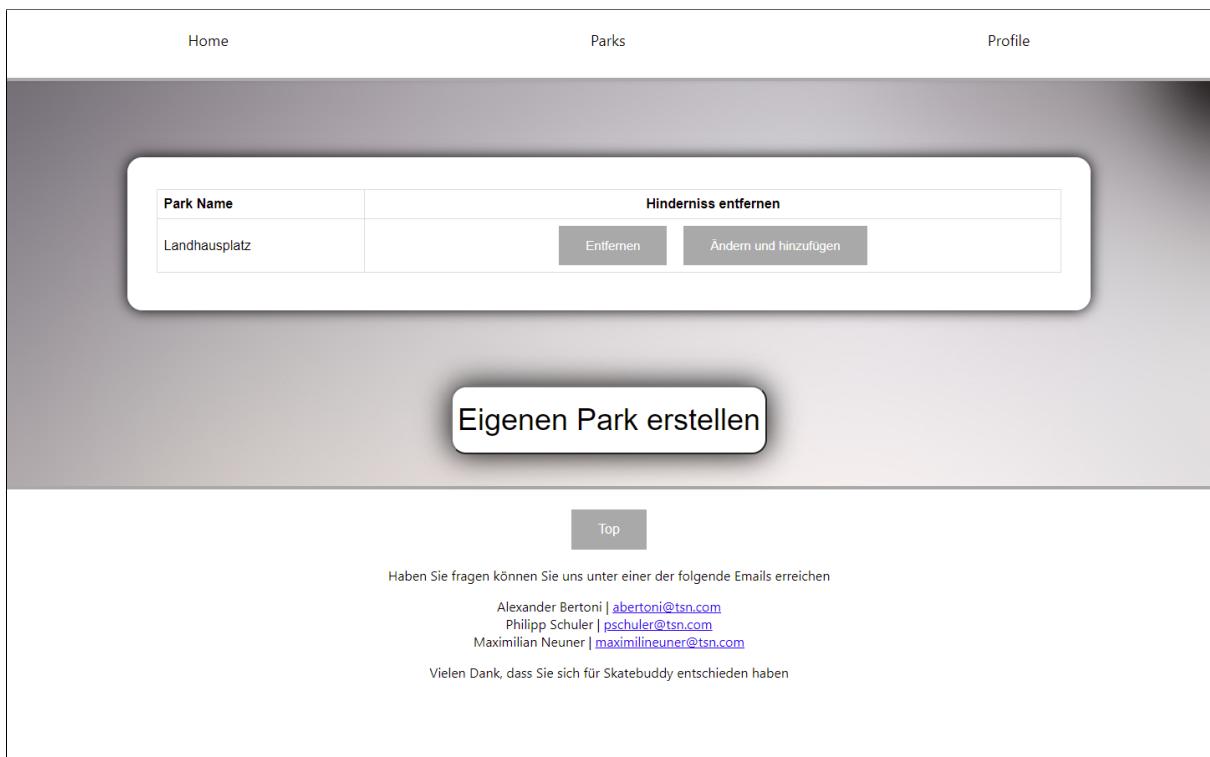


Abbildung 7.11.: Liste der Vorschläge

In dieser Ansicht kann der Admin alle erstellen Vorschläger der User erstellen. Er kann sie von hier aus löschen oder bearbeiten und als Park hinzufügen. Er kann von hier aus auch selbst einen komplett neuen Park hinzufügen, ohne einen Vorschlag zu bearbeiten.

7. Views

7.7.2. Park hinzufügen

The screenshot shows a modal window titled "Fügen Sie einen Park hinzu". It contains the following fields:

- Parkname: Landhausplatz
- Adresse: Landhausplatz 1, 6020 Innsbruck
- Busstop: Innsbruck Landhausplatz
- Koordinaten:
 - Latitude: 47
 - Longitude: 20
- Hindernisse:
 - Bank dropdown: Bank
 - Level dropdown: 1
 - Add button: Hinderniss hinzufügen
- A table for managing obstacles:

Hinderniss	Schwierigkeit	Hinderniss entfernen
Funbox	8	<button>Entfernen</button>
Quarter	8	<button>Entfernen</button>
- Save button: Park hinzufügen

Abbildung 7.12.: Park erstellen

Das Formular zum Park erstellen ist das Selbe wie dies zum Vorschläge erstellen. Benutzt der Admin einen Vorschlag welchen man bearbeiten möchte werden die Daten des Vorschlages in die einzelnen Felder geladen und können bearbeitet werden. Wurde der Vorschlag fertig bearbeitet und wird als Park der Datenbank hinzugefügt, wird der Vorschlag welcher bearbeitet wurde aus der Datenbank gelöscht. Wird kein Vorschlag benutzt sonder ein komplett neuer Park erstellt, sind die Input Felder von Anfang an leer.

7.8. Park Details

In den Details der einzelnen Parks findet man eine Map mit zwei Marker. Ein Marker für die Position des Parks und ein anderer f+r die derzeitige Position des Benutzers. Verweigert der Benutzer, dass auf seinen Standort zugegriffen werden darf, wird nur ein Marker an der Position des Parks platziert.

7. Views

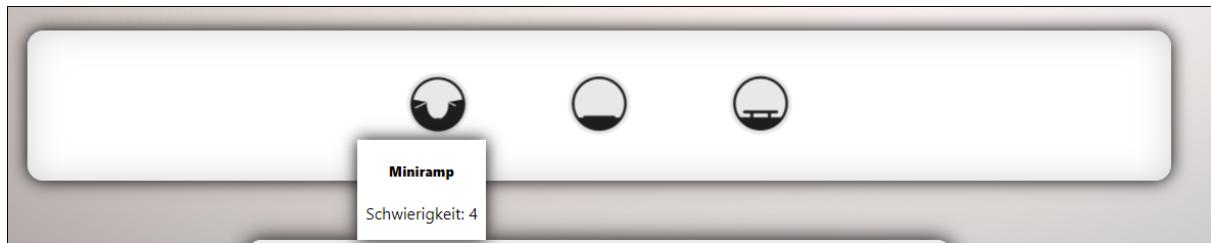


Abbildung 7.13.: Die Startseite

Unterhalb der Map stehen die Hindernisse welche sich in dem Park befinden. Schwebt man mit der Maus über eins der Hindernisse wird der Name und die Schwierigkeit des Hindernisses angegeben. Die Daten über das Hindernis und über den Park selbst, wird durch einen *useFetch* in Erfahrung gebracht. Ist man mit einem Benutzer angemeldet, kann man unter den Hindernissen eine Bewertung schreiben.

7. Views

7.8.1. Bewertungen



Abbildung 7.14.: Die Startseite

Damit es möglich ist eine Review zu schreiben muss ein Titel angegeben sein. Unter dem Titel steht der Benutzername des eingeloggten Users. Mit einem Klick auf den Sternen kann man dem Park eine Bewertung von eins bis fünf geben. Diese Sternenbewertung wurde mittels einer externen React Komponente names **React Star Ratings** von **ekeric13** gemacht. Drückt man auf den Senden Knopf wird die Review an den Server gesendet, welcher den dann in die Datenbank speichert. Während die Review an den Server gesendet wird, ist der Knopf deaktiviert, damit man bei wiederholten Knopfdruck nicht mehrere male eine Review an den Server sendet.



Abbildung 7.15.: Eine geschriebene Bewertung

Unter dem erstellen einer Bewertung kann man alle anderen Bewertungen dieses Parks sehen. Es ist möglich seine eigene Bewertung zu löschen. Dies erfolgt ganz einfach über die Überprüfung der BenutzerId mit der BenutzerId des Herstellers der Bewertung.

Teil IV.

Mobile App

8. Vorbereitung

8.1. Warum habe ich mich für React Native entschieden?

Im Sommer 2021 habe ich während meiner Arbeitszeit im GRZ IT Center in Innsbruck dazu entschieden, jeden Abend ein paar Stunden in das Lernen von neuen Technologien in Sachen Webentwicklung zu investieren. Als erstes auf meiner Liste stand der Kurs Modern JavaScript von NetNinja auf YouTube. NetNinja bietet auf seiner Webseite zu jedem freien Youtube-Kurs einen noch detaillierteren Kurs an, jedoch wollte ich kein Geld ausgeben, um zu lernen und offensichtlich macht es dann doch keinen großen Unterschied. Nach Modern JavaScript war klar, dass ich nun bereit war ein JavaScript- Framework zu lernen, um Benutzeroberflächen zu gestalten.

Durch Youtube hörte ich oft von den drei Frameworks React von Facebook, Angular von Google und Vue vom ehemaligen Google-Mitarbeiter Evan You. Ich wählte React aus, da es in Österreich und generell weltweit die meiste Anzahl von Jobs anbot. Nachdem ich React gelernt hatte und feststand, dass wir für unsere Diplomarbeit eine App benötigen, schlug ich sofort React Native als Cross-Platform-Lösung vor, damit ich mein bereits angeeignetes Wissen direkt anwenden kann.

Am 30. November 2021 begann ich offiziell mit der Entwicklung unserer Skater-App. Ich nutzte zeitweise das Projektmanagement-Tool Trello, um den Überblick über das Projekt zu behalten, es ist schließlich das bisher größte Projekt, an dem wir drei je gearbeitet haben. Im Nachhinein denke ich, es wäre viel besser gewesen, wenn ich sofort alles dokumentiert hätte und nicht jetzt, vier Monate später, mir alles noch einmal anschauen müsste.

9. Erstellung des Projekts

Schon bei der Erstellung des Projekts müssen einige Fragen geklärt werden. Als erstes sollte man sich wohl fragen, mit welcher CLI man das Projekt erstellen möchte.

Mit CLI (für engl. command-line-interface) ist in unserem Kontext eine Sammlung von Befehlen gemeint, die in der Kommandozeile eines PCs ausgeführt werden können. Beispiele für Kommandozeilen- Emulatoren:

- Windows Powershell
- Windows Cmd
- MacOS zsh
- Git Bash
- Gnome Shell

Um ein React Native Projekt zu erzeugen kommen diese CLIs in Frage:

- React Native CLI
- Expo CLI
- IgniteCLI [32]
- Create-React-Native-App [33]

9. Erstellung des Projekts

9.1. React Native CLI

Wichtig: In der Installationsanleitung auf der Webseite von React Native kann man sein aktuelles Betriebssystem und Zielplattform auswählen. Apple erlaubt nicht die Entwicklung von Apps auf anderen Betriebssystemen als MacOS selbst, d.h. die Entwicklung der iOS-App ist auf Windows 11, meinem Betriebssystem, nicht möglich. Ich hatte während der Entwicklung nie Zugriff auf einen PC mit MacOS, daher wurde die gesamte App ausschließlich für Android entwickelt. Doch sollten wir jemals diese App wirklich veröffentlichen wollen, so wird das Umschreiben keinen großen Aufwand darstellen, da wir immer darauf geachtet haben, nur Bibliotheken zu verwenden, die auch für iOS kompatibel sind.

9.1.1. Abhängigkeiten und Erstellung des Projekts

Für die Verwendung der React-Native-CLI werden einige andere Softwarepakete benötigt, darunter – natürlich – Node.js, mindestens Version 12. Außerdem wird benötigt:

- Java SE Development Kit (mind. Version 11)
- Android Studio
 - Android SDK 11 (R)
 - Android SDK Platform: API Level 30
 - Android Virtual Device: Google Pixel 2 mit Android 11

Sobald alle Abhängigkeiten installiert wurden, führt man folgenden Befehl aus, um ein neues React Native Projekt zu erstellen:

```
1 C:\example> npx react-native init reactNativeInit
```

Code-Snippet 9.1.: CMD - React Native CLI init

NPX ist ein Befehl, der seit Version 5.2.0 in NPM enthalten ist. Er ist speziell bei CLIs hilfreich, denn anstatt das Paket react-native auf dem PC zu installieren und dann aufzurufen, wird mit dem Befehl automatisch die neueste Version der CLI von einem Server abgefragt und ausgeführt. So kann man sicherstellen, dass niemals eine veraltete Version der CLI verwendet wird.

9. Erstellung des Projekts

9.1.2. Ordnerstruktur

9.1.2.1. Ordner

- **__tests__/:**

In diesem Ordner werden Skripts abgespeichert, die die Funktionalität der App überprüfen sollen, sogenannte Tests. React Native empfiehlt hierbei die Bibliothek Jest, welche die Durchführung von Snapshot-Tests ermöglicht. Diese testen, ob sich die Benutzeroberfläche auch so verhältet, wie erwartet. Tests sind in großen Projekten nicht mehr wegzudenken, bei kleineren Projekten sind sie aber meist den Schreibaufwand nicht wert.

- **.vscode/:**

In diesem Ordner ist es möglich Einstellungen und Skripts für den Text-Editor Visual Studio Code abzuspeichern, den wir durchwegs für die gesamte Diplomarbeit verwendet haben.

- **android/:**

Manche Änderungen kann man nicht mit Hilfe von JavaScript vornehmen, manchmal muss man auch die plattformspezifischen Dateien der App verändern. Diese sind alle im Ordner android vorhanden. In ihm sind hauptsächlich Konfigurationsdateien für das Build-Automatisierungstool Gradle und Manifest- Dateien, welche Metadaten zur App enthalten und auch für andere Bibliotheken bearbeitet werden müssen, z.B. Google-Maps-Funktionalität.

- **node_modules/:**

Im Ordner node_modules werden alle Bibliotheken abgespeichert, die mit Hilfe des Paketmanagers in das Projekt eingebunden wurden.

- **bundle/, ios/:**

Hier befinden sich lediglich Dateien, welche für das Entwickeln der iOS-App wichtig sind. Bei unserer Diplomarbeit wurden diese Ordner nicht benötigt.

9. Erstellung des Projekts

9.1.2.2. Dateien

- **index.js, App.js:**

index.js ist, wie der Name schon vermuten lässt, die Einstiegsdatei für das Projekt. In ihr wird nur die Datei App.js eingebunden.

App.js ist also die erste richtige Code-Datei, welche von uns bearbeitet wird. Als Erklärung zeige ich hier ein klassisches Hallo-Welt-Programm, welches eigentlich von der Expo-CLI erstellt wird, jedoch sehr gut zur Veranschaulichung dient.

```
1 import { StyleSheet, Text, View } from 'react-native';
2
3 function App() {
4   return (
5     <View style={styles.container}>
6       <Text>Open up App.js to start working on your app!</Text>
7     </View>
8   );
9 }
10
11 const styles = StyleSheet.create({
12   container: {
13     flex: 1,
14     backgroundColor: '#fff',
15     alignItems: 'center',
16     justifyContent: 'center',
17   },
18 });
19
20 export default App;
```

Code-Snippet 9.2.: React Component - App.js

In der ersten Zeile des Programms werden diverse React Native Core Components importiert. Core Components sind, wörtlich übersetzt, die Kernkomponenten des Frameworks, mit ihnen wird der Großteil der Benutzeroberfläche aufgebaut. Wie bereits erwähnt, werden die React Native Komponenten beim Kompilieren in die richtigen Komponenten für das Zielsystem umgewandelt.

9. Erstellung des Projekts

REACT NATIVE UI COMPONENT	ANDROID VIEW	IOS VIEW	WEB ANALOG	DESCRIPTION
<code><View></code>	<code><ViewGroup></code>	<code><UIView></code>	A non-scrolling <code><div></code>	A container that supports layout with flexbox, style, some touch handling, and accessibility controls
<code><Text></code>	<code><TextView></code>	<code><UITextView></code>	<code><p></code>	Displays, styles, and nests strings of text and even handles touch events
<code><Image></code>	<code><ImageView></code>	<code><UIImageView></code>	<code></code>	Displays different types of images
<code><ScrollView></code>	<code><ScrollView></code>	<code><UIScrollView></code>	<code><div></code>	A generic scrolling container that can contain multiple components and views
<code><TextInput></code>	<code><EditText></code>	<code><UITextField></code>	<code><input type="text"></code>	Allows the user to enter text

Abbildung 9.1.: Tabelle - Core Components und deren Gegenstücke [34]

In der nächsten Zeile wird unserer erster React-Component erzeugt, welcher im Grunde nur eine Funktion ist, die JSX-Code als Rückgabewert liefert.

In Zeile 5 wird zur View ein React Native StyleSheet zugewiesen. Man verwendet nämlich kein gewöhnliches CSS, wie in der Webentwicklung, sondern ein relativ ähnlich aufgebautes, eigenes System zur Gestaltung der App. Ein wichtiger Unterschied ist, dass die Attribut-Namen im StyleSheet nicht durch Bindestriche getrennt, sondern in der LowerCamelCase-Notation geschrieben werden [35].

Am Ende der Datei wird noch die Komponente als Default exportiert, damit sie von Expo verarbeitet werden kann [36].

- **app.json, package.json:**

app.json wird von React Native generiert und enthält zu Beginn bloß den Namen und Anzeigennamen der App.

package.json hat einen ähnlichen Zweck, auch darin sind Name, Versionsnummer, aber auch der Eintrittspunkt in die App abgespeichert. In Scripts werden Kommandozeilenbefehle abgespeichert, die mit Hilfe des Befehls `npm run jscript` aufgerufen werden können. Im nächsten Objekt werden die Namen der Bibliotheken aufgelistet, welche für das Ausführen der App benötigt werden. Bibliotheken, die nur während der Entwicklung benötigt werden, sind unter devDependencies abzuspeichern (beim Installieren den Tag `--save-dev` anhängen).

- **package-lock.json:**

package-lock.json gehört zu NPM und ist eine Lockfile. In einer Lockfile werden alle installierten Dependencies noch einmal aufgelistet und mit der Versionsnummer abgespeichert, um sicherzugehen, dass bei der Generierung von `node_modules/` alle Bibliotheken immer dieselbe Version haben.

- **babel.config.js:**

Babel ist ein sogenannter JavaScript-Transpiler. Er wandelt JavaScript-Features, welche

9. Erstellung des Projekts

erst in späteren Versionen des ECMAScript-Standards eingebaut und möglicherweise noch nicht von allen Systemen unterstützt werden, in JavaScript-Code um, welcher auch von älteren Versionen verstanden werden kann, um (vgl. ein Compiler wandelt lesbaren Code in Maschinencode um). Außerdem hat er noch einige andere Funktionen, welche durch Plugins hinzugefügt werden können. In der Konfigurationsdatei wird nur eine vorgefertigte Liste an Plugins in das Projekt eingebunden, das "babel-preset-expo".

- **metro.config.js:**

Eine Konfigurationsdatei für den JavaScript-Bundler Metro.

- **.gitignore:**

In die Datei .gitignore fügt man Datei- und Ordnernamen ein, die nicht von der Versionsverwaltung gesichert werden sollen. Sehr zu empfehlen ist dies beim Ordner node_modules, da dieser schon direkt nach der Erstellung 170 Megabytes groß ist und er aus den Dateien package.json und yarn.lock generiert werden kann.

- **.eslintrc.js:**

ESLint ist ein Linter für JavaScript, er überprüft alle vorhandenen JavaScript-Dateien nach möglichen syntaktischen oder semantischen Fehlern und kann sogar mögliche Verbesserungen vorschlagen.

- **.prettierrc.js:**

Prettier ist ein Werkzeug, welches jeder Programmierer verwenden sollte. Es übernimmt die Aufgabe des Formatierens von Quellcode, sodass ein einheitliches System herrscht und Zeit gespart wird.

- **.flowconfig:**

In JavaScript gibt es, Stand 2022, keine statische Typsicherheit bei Variablen, jede Variable kann jeden Typ annehmen. Flow ist ein System, welches in Kommentaren in Code die Typen von bestimmten Variablen festlegt und beim Schreiben der App Fehler anzeigt, falls diese verletzt werden. Stattdessen könnte man auch die Sprache TypeScript verwenden, welche zu gewöhnlichem JavaScript kompiliert werden kann.

- **.watchmanconfig:**

Watchman wurde von Facebook entwickelt, um die Veränderung von Dateien zu registrieren und bestimmte Prozesse dadurch zu starten, z.B. das Kompilieren von TypeScript zu JavaScript.

- **Gemfile, Gemfile.lock:**

Diese beiden Dateien werden nur bei der Entwicklung von iOS-Apps benötigt. Sie gehören zum Ruby- Projekttool Bundler.

9. Erstellung des Projekts

9.1.3. Metro

Beim Start der App wird als erstes der Metro Bundler initialisiert. Er besteht aus einem Server, welcher den geschriebenen Code mittels Babel kompiliert und anschließend an die App schickt. Dadurch verhindert man, jedes mal die App neu auf das Gerät installieren zu müssen, um eine kleine Änderung im Text sehen zu können. Beim Einbinden von neuen JavaScript-Bibliotheken muss allerdings der Bundler neu gestartet werden, um diese verwenden zu können.

Um den Server zu starten gibt man folgenden Befehl ein:

```
1 C:\example\reactNativeInit> npm start
```

Code-Snippet 9.3.: CMD - Kurzschreibweise für den Befehl npm run start

The screenshot shows a terminal window with the following text:
\$ npm start
> mobile@0.0.1 start
> npx react-native start

A large decorative graphic made of '#' characters follows.

Welcome to Metro!
Fast - Scalable - Integrated

To reload the app press "r"
To open developer menu press "d"

BUNDLE ./index.js

LOG Running "mobile" with {"rootTag":1}

Abbildung 9.2.: Der Metro-Server in Aktion

9. Erstellung des Projekts

Als nächstes öffnet man den Ordner root/android in Android Studio. Man hat nun zwei Möglichkeiten, die App auszuführen:

1. Mit einem virtuellen Android Gerät (Android Virtual Device AVD):

Dazu öffnet man in Android Studio den AVD-Manager und erstellt einen neuen Emulator mit einer Android Version von 11. Danach startet man den Emulator und führt folgenden Befehl aus:

```
1 C:\example\reactNativeInit> npm run android
```

Code-Snippet 9.4.: CMD - Die App wird auf dem Emulator installiert und ausgeführt.

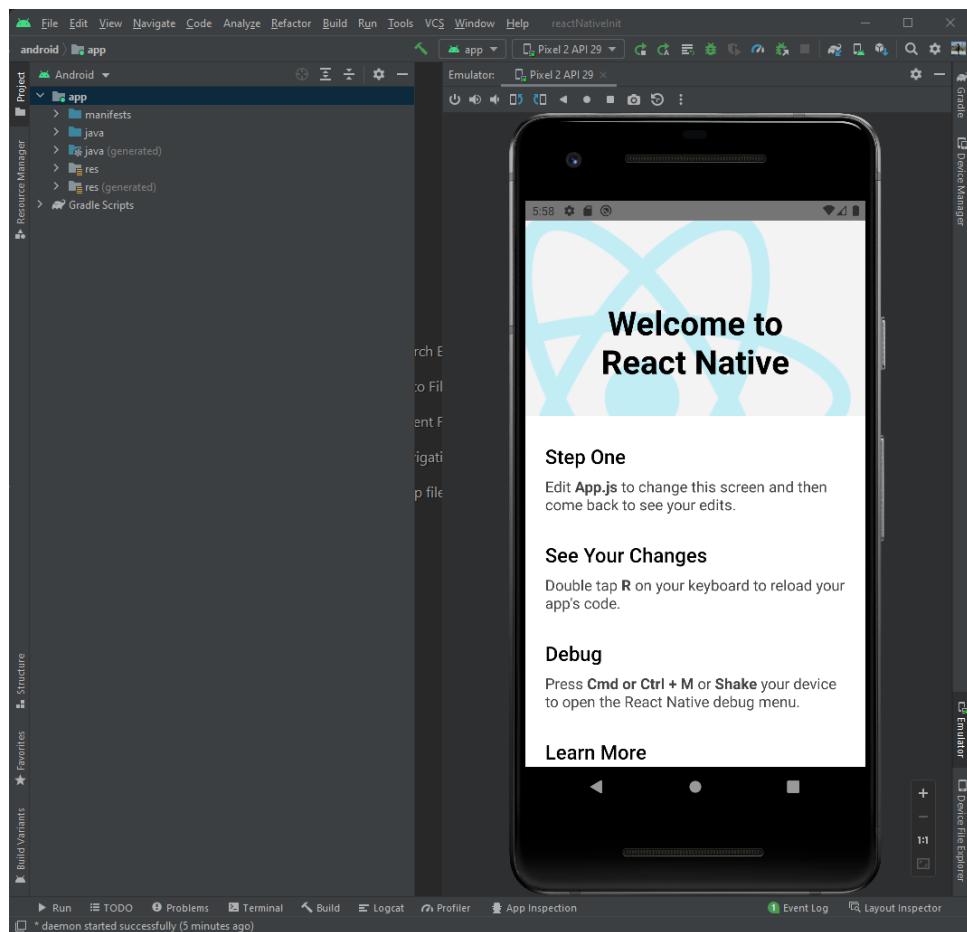


Abbildung 9.3.: Die Beispiel-App in Android Studio

9. Erstellung des Projekts

2. Mit einem physischen Android-Gerät:

Als erstes aktiviert man auf dem Gerät in den Einstellungen USB-Debugging und schließt es per USB an den PC an. Wenn alle Treiber installiert sind, müsste es dann direkt in Android Studio erkannt werden, um die App zu installieren.

Jetzt muss noch ein Tunnel über die USB-Verbindung erzeugt werden, damit das Gerät darüber mit dem lokalen Metro-Server kommunizieren kann. Man listet alle Geräte auf und erzeugt dann mit der Device-Identifikation einen Tunnel.

```
1 C:\example\reactNativeInit> adb devices
2 List of devices attached
3 3429905j850496g3          device
4 emulator-8125              device
5
6 C:\example\reactNativeInit> adb -s 3429905j850496g3 reverse tcp:8081
    tcp:8081
7 8081
```

Code-Snippet 9.5.: CMD - Erstellung eines TCP-Tunnels über USB

9. Erstellung des Projekts

9.2. Expo CLI

Die Entwickler von React Native selbst empfehlen allen Anfängern im Gebiet App-Entwicklung die Verwendung der Expo CLI [37], welche eine vereinfachte Variante einer React Native Anwendung erzeugt.

Als Vorteil zählt auf jeden Fall die Geschwindigkeit, mit der eine neue App auf einem neuen Gerät getestet werden kann. Dies ist meist innerhalb weniger Minuten möglich.

Ein wichtiger Nachteil ist jedoch, dass man in einem Expo-Projekt eingeschränkten Zugriff auf Schnittstellen des Betriebssystems hat, es sind im Projekt nicht einmal die Ordner android und ios vorhanden, um Änderungen vorzunehmen.

9.2.1. Installation und Erstellung einer Expo App

Um eine Expo React Native Anwendung erstellen zu können benötigt man als erstes Node.js und den darin enthaltenen Node Package Manager. In einer Kommandozeile führt man nun folgende Befehle aus, um die Expo-CLI im globalen Kontext zu installieren und anschließend ein Expo Projekt zu erstellen. Nach der Auswahl für die Vorlage wird das Projekt erstellt.

```
1 C:\example> npm install -g expo-cli
2 added 1549 packages, and audited 1550 packages in 1m
3
4 C:\example>expo init expoInitBlank
5 ? Choose a template: — Use arrow-keys.
6     _____ Managed workflow _____
7 >     blank                 a minimal app as clean as an empty canvas
8     blank (TypeScript)
9     tabs (TypeScript)
10    _____ Bare workflow _____
11    minimal
```

Code-Snippet 9.6.: CMD - Als Beispiel-Projekt ist das Template blank geeignet.

- Ordner:

- **.expo-shared, .git:**

In den Ordnern .expo-shared, .git und .vscode befinden sich lediglich Konfigurationsdateien für Expo selbst und die Versionsverwaltungssoftware Git.

- **assets:**

Der Ordner assets ist für das Abspeichern von statischen Ressourcen, wie Bildern, Icons oder ähnliches gedacht.

9. Erstellung des Projekts

- Dateien:

- **app.json**

In app.json werden bei der Erstellung mit Expo zusätzliche Informationen zur App abgespeichert, darunter auch die Versionsnummer, Bildschirmausrichtung und noch viele weitere Optionen.

- **yarn.lock**

Expo erzeugt ein Blank-Template standardmäßig mit Yarn, man kann aber auch stattdessen mit dem Tag --npm bei der Projekterstellung NPM einbinden.

Die App kann nun sofort getestet werden, als erstes startet man wieder den JavaScript-Bundler Metro mit folgendem Befehl:

```
$ expo start
Starting project at C:\example\expoInitBlank
Developer tools running on http://localhost:19002
Opening developer tools in the browser...
Starting Metro Bundler

> Metro waiting on exp://10.0.0.21:19000
> Scan the QR code above with Expo Go (Android) or the Camera app (iOS)

> Press a | open Android
> Press w | open web

> Press r | reload app
> Press m | toggle menu
> Press d | show developer tools
> shift+d | toggle auto opening developer tools on startup (enabled)

> Press ? | show all commands

Logs for your project will appear below. Press Ctrl+C to exit.
ios Bundling complete 2477ms
ios Running app on iPhone 8 PS

```

Abbildung 9.4.: Expo beim Start

9. Erstellung des Projekts

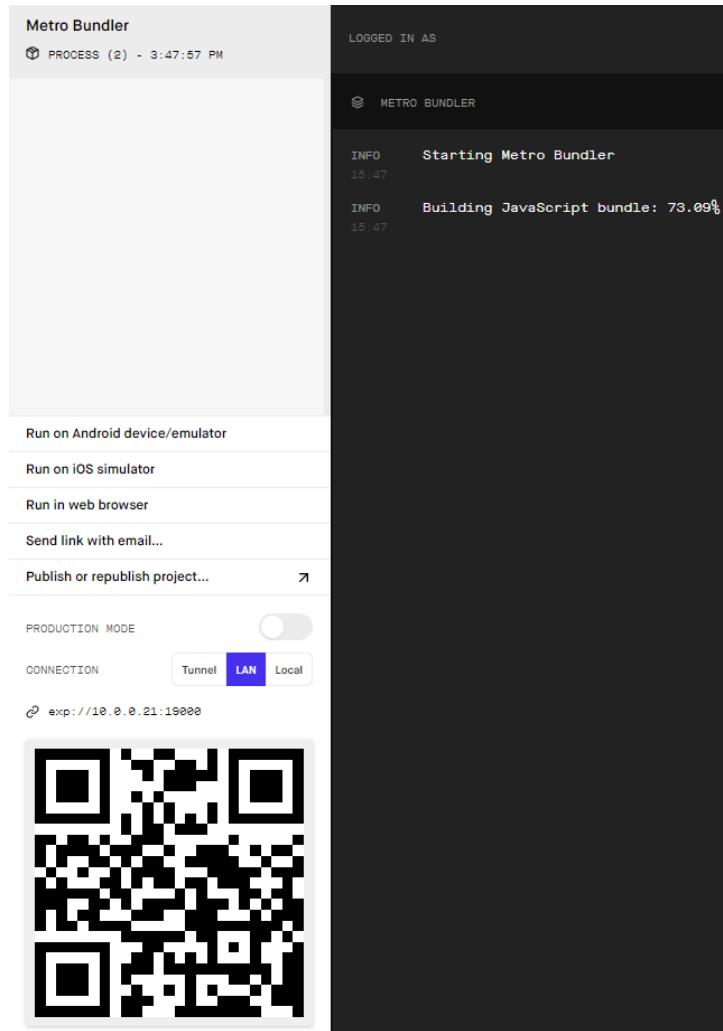


Abbildung 9.5.: Webinterface von Expo

Nach dem Start wird im Webbrowser ein Entwickler-Menü geöffnet, auf dem noch einmal der QR-Code und noch einige weitere hilfreiche Funktionen zu finden sind.

Um nun die App zu testen, muss man noch die ExpoGo App auf seinem Mobiltelefon installieren und den gezeigten QR-Code scannen, während man im selben Netzwerk ist. Danach wird eine Bundle-Anfrage an den lokalen Metro-Server geschickt, er kompiliert unser Projekt und schickt es an den Client.

9. Erstellung des Projekts

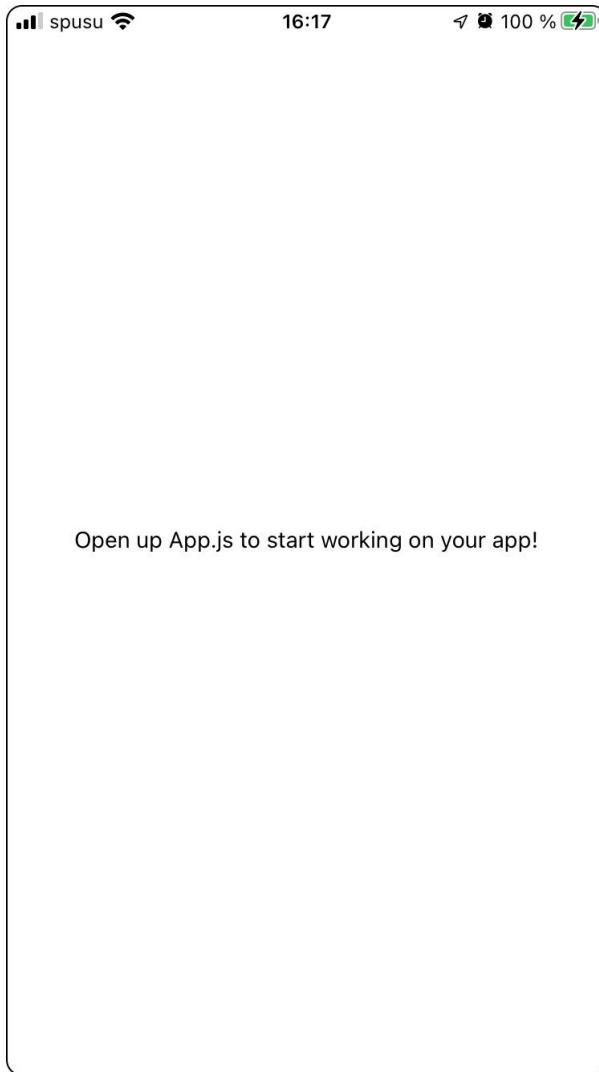


Abbildung 9.6.: Die weiße Leinwand von Expo

9.2.1.1. Expo Eject

Wie bereits vorher schon erwähnt, hat es einige Nachteile seine App mit Expo zu machen. Um aber einen schnellen Prototypen zu bauen ist das Framework perfekt. Um also eine Expo App in eine vollwertige React Native Anwendung umzuwandeln, benötigt man folgenden Befehl.

```
1 C:\example\expoInitBlank> expo eject
```

Code-Snippet 9.7.: CMD - Man wird bei einer Eingabeaufforderung darüber informiert, dass dieser Prozess nicht rückgängig gemacht werden kann.

10. Aufbau

Die Benutzeroberfläche wird durch Screens, Components und StyleSheets zusammengesetzt. Screens sind komplette Ansichten, sie belegen den ganzen Bildschirm des Geräts. Alle Elemente in einem Screen werden durch Components aufgebaut. Den Feinschliff erledigen die StyleSheets, in ihnen werden alle Informationen gespeichert, welche für die Positionierung, Größe, Farbe, usw. von Komponenten wichtig sind.

10.1. Screens

In unserer App wollen wir Informationen für den Benutzer verständlich darstellen. Es würde keinen Sinn machen alle Informationen auf die gleiche Seite zu schreiben, eine logische Gliederung der App ist daher sehr wichtig. Screens sind im Grunde nur React-Komponenten; sie werden aus mehreren kleineren Komponenten zusammengebaut.

10.2. Arten von Screens

In Apps gibt es eigentlich nicht so viele Arten von Screens, hier die häufigsten Anwendungsfälle im Überblick:

10.2.1. Listen

Listen-Screens sind dafür da, um mehrere Datensätze untereinander oder nebeneinander darzustellen. Dafür wird über jedes Element in einer Liste iteriert und an eine Komponente weitergegeben, welche anschließend die Daten aufbereitet und darstellt.

10.2.2. Details

Nachdem man jedes Element aufgelistet hat, sollte es noch möglich sein ein Element genauer zu betrachten bzw. alle Informationen anzuzeigen, welche nicht im Überblick der Liste Platz haben.

10.2.3. Formulare

In Formularen kann der Benutzer Informationen in die App eintragen, welche dann per HTTP an das Backend geschickt werden.

10.3. Komponenten

Komponenten, die in vielen anderen Komponenten gebraucht werden, werden bei uns im Ordner root/components/common gespeichert. Andere Komponenten werden im components-Ordner nach ihrem zugehörigen Screen getrennt.

10.3.1. Snippets

Um Komponenten schneller erzeugen zu können gibt es in Visual Studio Code Snippets, welche immer den selben Code generieren.



Abbildung 10.1.: Beispiel für ein Snippet

The screenshot shows the generated code in a code editor. It consists of a single line of code: 'import React from 'react''. The word 'React' is highlighted in red, indicating it is a variable or identifier.

Abbildung 10.2.: Vom Snippet erzeugter Code

Da jede Komponente in unserer App ähnlich aufgebaut ist und damit wir nicht jedes mal den gleichen Code schreiben müssen, habe ich ein eigenes Snippet erstellt. Die Abkürzung rnfec steht für React Native Functional Component Export Custom und erzeugt folgenden Code:

```

1 import React from 'react'
2 import { View } from 'react-native'
3 import Text from '@components/common/Text';
4
5 const Test = () => {
6   return (
7     <View>
8       <Text></Text>
9     </View>
10   )
11 }
12
13 export default Test

```

Code-Snippet 10.1.: React Component - Snippet rnfec in der Datei Test.js

Der Name der Komponente wird hierbei vom Dateinamen abgeleitet. Eine Besonderheit, die auffällt, ist das Import-Statement von Text. Das Verwenden von Alias-Paths wird durch das Babel-Plugin "module-resolver" möglich gemacht.

10.3.2. Text

In React Native gibt es keinen sicheren Weg, um allen Text-Komponenten automatisch eine Font zuzuweisen, deswegen wird in unserer App immer eine eigene Text-Komponente verwendet. Diese wird auch beim Erzeugen von neuen Komponenten automatisch eingebunden, damit nicht versehentlich doch die falsche Komponente verwendet wird.

```
1 import colors from '../styles/Colors';
2
3 const CustomText = ({ style, children }) => (
4   <Text style={[styles.customFont, style]}>{children}</Text>
5 );
6
7 const styles = StyleSheet.create({
8   customFont: {
9     fontFamily: 'Poppins-Regular',
10    color: colors.text,
11  },
12});
13
14 export default CustomText;
```

Code-Snippet 10.2.: React Component - Custom Text Komponente

Alle Kinder dieser Komponente werden dann direkt in den Text eingebunden und, sollte man noch ein style-Property an die Komponente übergeben wird der Standard-Style überschrieben.

10.3.3. Button

```

1 import React from 'react';
2 import { Pressable } from 'react-native';
3 import MaterialCommunityIcons from 'react-native-vector-icons/
  MaterialCommunityIcons';
4
5 import Text from './Text';
6 import styles from '@styles/GlobalStyles';
7
8 const Button = ({
9   onPress,
10  title,
11  icon,
12  iconType,
13  style,
14  iconSize = 24,
15 }) => (
16   <Pressable
17     onPress={onPress}
18     style={[styles.buttonContainer, styles.shadow, style]}>
19     >
20       <Text style={styles.buttonText}>{title}</Text>
21
22       {icon && iconType === 'mci' && (
23         <MaterialCommunityIcons
24           name={icon}
25           size={iconSize}
26           color={styles.buttonText.color}
27         />
28       )}
29       ...
30     </Pressable>
31   );
32
33 export default Button;

```

Code-Snippet 10.3.: React Component - Auch der Button-Component von React Native lässt sich leicht nachbauen.

Die Funktion, welche beim Drücken aufgerufen werden soll, wird der Komponente als onPress übergeben. Sollte das Prop icon leer sein, wird kein Icon gerendert, andernfalls wird noch überprüft, welche Art von Icon verwendet werden soll. Die Icons werden nämlich von mehreren Anbietern bereitgestellt.

10. Aufbau

10.3.4. TextInput

Auch die Komponente TextInput wird von uns überschrieben. Sie besteht zusätzlich immer aus einem Icon auf der linken Seite. Je nach dem, ob der Input gerade bearbeitet wird, wird die Farbe verändert. Sollte bei der Textüberprüfung ein Fehler auftreten, so wird die Komponente rot gefärbt.

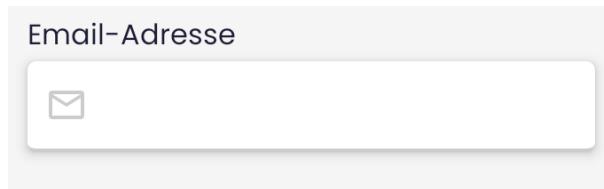


Abbildung 10.3.: Der TextInput für die Email am Login-Screen



Abbildung 10.4.: TextInput ausgewählt



Abbildung 10.5.: Fehlerhafte Eingabe

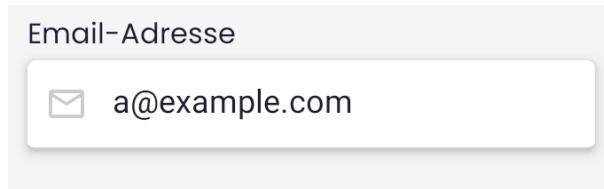


Abbildung 10.6.: Korrekte Eingabe

10.4. Styles

10.4.1. Schriftart

Durch den Common-Component Text können wir jedem Text in der App eine einheitliche Schriftart geben. Ich entschied mich für die Font Poppins [38].

10.4.2. Farbwerte

Alle Farben, die man in der App sehen kann, wurden in der Datei Colors.js in root/styles festgelegt. So muss nicht, sollte sich das Design ändern müssen, in jedem StyleSheet jede Farbe einzeln getauscht werden.

Die Hauptfarbe der App ist der wunderschöne Grünton Jade (#00A86B), welchen ich durch die Webseite color-name.com [39] fand. Die Webseite MyColor.Space [40] erzeugt anhand einer Farbe mehrere andere Farben, die gut dazu passen.

10.4.3. Icons

Die Bibliothek react-native-vector-icons [41] war eine große Hilfe bei der Erstellung des Designs. Icons sind nämlich der perfekte Weg, um Informationen in wenig Platz darzustellen, z.B. welche Funktion ein Knopf hat. Ich habe stets darauf geachtet diverse Icons von verschiedenen Anbietern in die App einzubauen. Hauptsächlich verwendete ich die MaterialCommunityIcons, eine große Sammlung von Icons orientiert am Material Design, einer Designsprache entwickelt von Google.

10.4.4. FlashMessage

```
1 const App = () => (
2   ...
3   <FlashMessage position="top" />
4   ...
5 );
6
7 export default App;
```

Code-Snippet 10.4.: React Component - FlashMessage-Komponente in root/App.js

Die Komponente FlashMessage ist für Mitteilungen über Geschehnisse in der App zuständig, z.B. Login fehlgeschlagen oder Login erfolgreich.

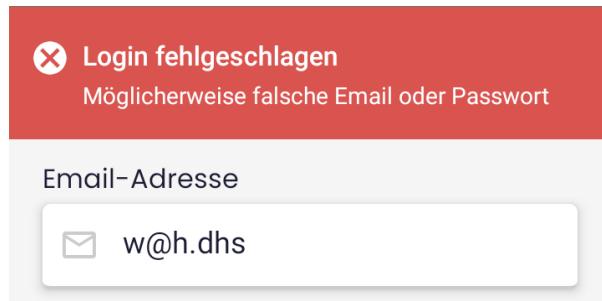


Abbildung 10.7.: Login-Informationen falsch

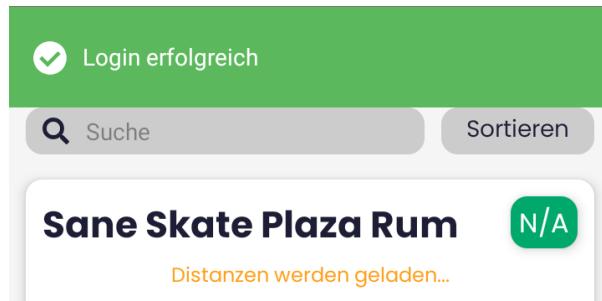


Abbildung 10.8.: Login-Informationen richtig

11. Hooks

In unserer App kommen drei selbst verfasste Hooks zum Einsatz.

11.1. useFetch

Der useFetch-Hook wurde bereits in der Webseite verwendet, die App nutzt eine leicht abgewandelte Version, damit man auch die Daten aktualisieren kann. Mehr dazu im Abschnitt Daten bekommen.

```
1 ...
2 const [ refresher , setRefresher ] = useState( false );
3
4 const refreshData = () => {
5     setRefresher( prevRefresher => !prevRefresher );
6 };
7
8 useEffect( () => {
9     ...
10 } , [ url , refresher ] );
11 ...
```

Code-Snippet 11.1.: JavaScript Funktion - useFetch mit Aktualisierungsfunktion

useEffect ist ein React-Hook. Er führt seine übergebene Funktion dann aus, wenn sich eine Variable in der im zweiten Argument übergebenen Liste ändert. Sollte die Liste leer sein, so wird die Funktion nur ausgeführt, sobald die Komponente das erste Mal gerendert wird.

Sobald die Variable refresher nun also geändert wird, wird die Funktion im useEffect-Hook neu ausgeführt.

11. Hooks

11.2. useDirections

Der Hook `useDirections` ist für die Google Maps Directions API-Anfragen zuständig. Google bietet über ihre API die Möglichkeit an, die Distanz zwischen zwei Orten abzufragen, genau wie in Google Maps.

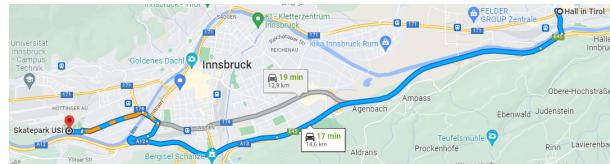


Abbildung 11.1.: Distanz zwischen Hall in Tirol und Skatepark USI Innsbruck

```
1 const buildUrl = method => {
2   let url = 'https://maps.googleapis.com/maps/api/directions/json?
3     origin=';
4   url += location.coords.latitude.toString();
5   url += ',';
6   url += location.coords.longitude.toString();
7   url += '&destination=';
8   url += skatepark.latitude.toString();
9   url += ',';
10  url += skatepark.longitude.toString();
11  url += '&mode=';
12  url += method;
13  url += '&key=';
14  url += secrets.apiKey;
15  return url;
```

Code-Snippet 11.2.: JavaScript Funktion - Die Url für den API-Request wird generiert.

Mit dieser Funktion bauen wir uns die URL für die Anfrage zusammen. Als Method übergibt man den String "walking", "bicycling", "transit" oder "driving".

11.2.1. Api-Key

Um diese API benutzen zu können benötigt man einen API-Schlüssel, welchen man unter <https://console.cloud.google.com> erhält. Google ist mit ihrer Cloud Console einer der größten Anbieter von Cloud Computing neben Amazon AWS und Microsoft Azure. Für diese Anfragen benötigt man Zugriff auf die "Directions API".

11.3. useLocation

In diesem Hook wird die aktuelle GPS-Position vom Gerät abgefragt. Der Benutzer wird, sofern er dies noch nicht getan hat, nach der Erlaubnis gefragt, die genaue Position des Gerätes zu ermitteln und mit der App zu teilen. Dies wird benötigt, um die Distanz zwischen Benutzer und Skateparks zu berechnen.

In Zeile 3 wird mit Hilfe der PermissionsAndroid-API, bereitgestellt von React Native, überprüft ob die Erlaubnis schon erteilt wurde, die Position zu ermitteln.

```

1 const checkPermission = async () => {
2   if (Platform.OS === 'android') {
3     const status = await PermissionsAndroid.check(
4       PermissionsAndroid.PERMISSIONS.ACCESS_FINE_LOCATION,
5     );
6
7     if (status) {
8       setLocError(null);
9       return true;
10    }
11  }
12  return false;
13};

```

Code-Snippet 11.3.: JavaScript Funktion - Berechtigung prüfen.

Sollte dies nicht der Fall sein, so wird die Erlaubnis angefragt.

```

1 const requestPermission = async () => {
2   if (Platform.OS === 'android') {
3     const response = await PermissionsAndroid.request(
4       PermissionsAndroid.PERMISSIONS.ACCESS_FINE_LOCATION,
5     );
6     if (response === PermissionsAndroid.RESULTS.GRANTED) {
7       setLocError(null);
8       return true;
9     }
10   }
11   setLocError('Standortdienst wurde abgelehnt');
12   return false;
13};

```

Code-Snippet 11.4.: JavaScript Funktion - Berechtigung anfragen.

11. Hooks

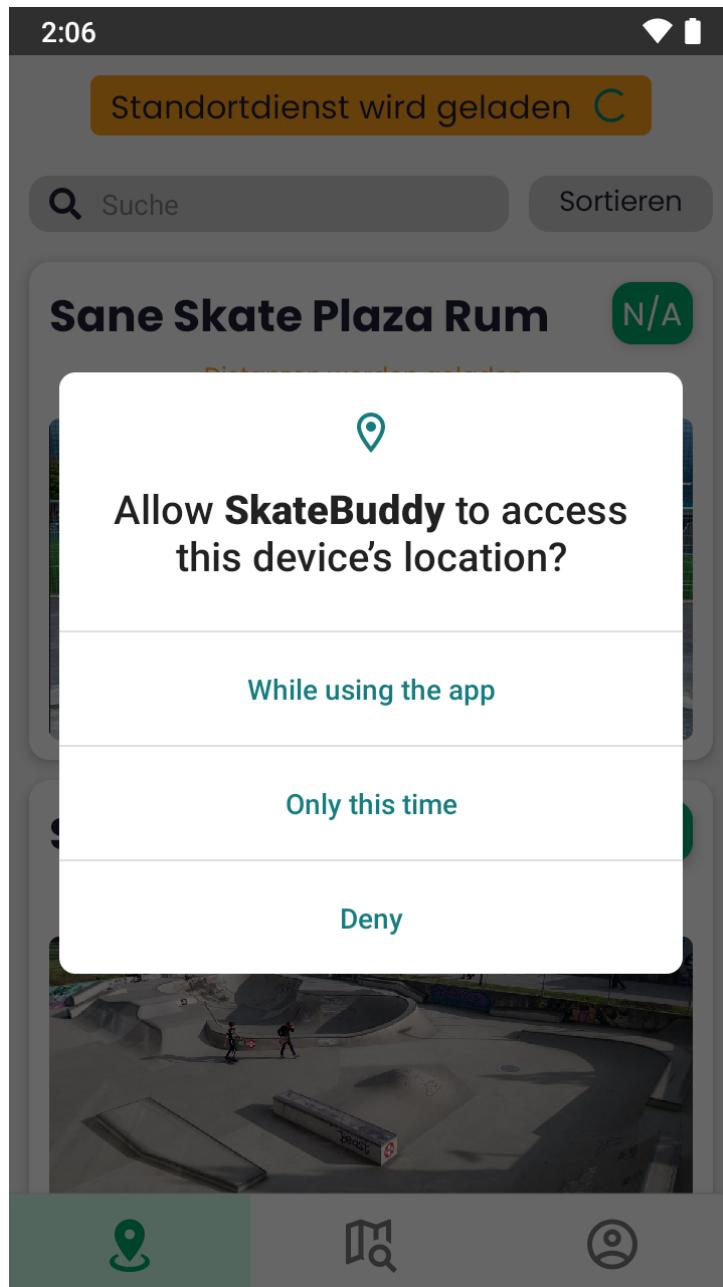


Abbildung 11.2.: Die Berechtigungs-Anfrage beim Starten der App

11. Hooks

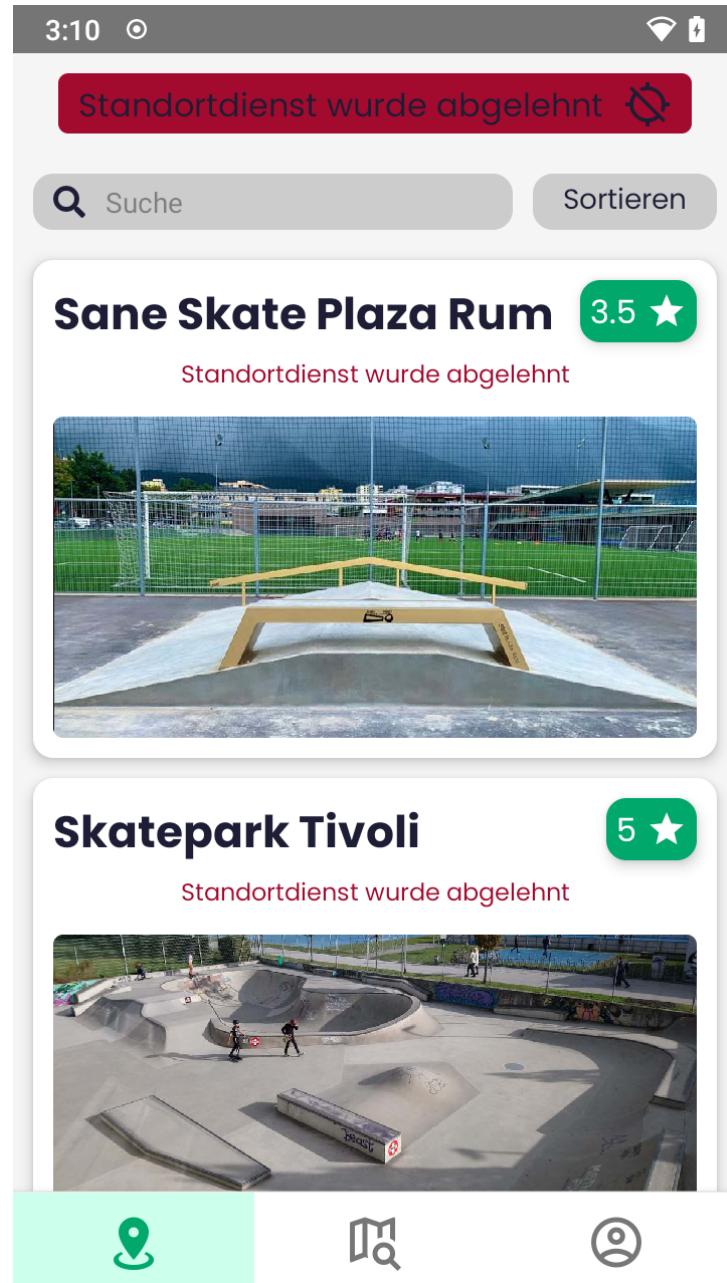


Abbildung 11.3.: Beim Ablehnen der Aufforderung wird die Anfrage an die Directions-API abgebrochen

Wenn der Benutzer den roten Knopf drückt, wird er noch einmal gebeten die Berechtigung zu erteilen.

11. Hooks

```
1 const getLocation = async () => {
2     setLocLoading(true);
3     setLocError(null);
4     if ((await checkPermission()) || (await requestPermission())) {
5         Geolocation.getCurrentPosition(
6             geolocation => {
7                 setLocation(geolocation);
8                 setLocLoading(false);
9             },
10            error => {
11                setLocError(`#${error.code} ${error.message}`);
12                setLocLoading(false);
13            },
14            { enableHighAccuracy: true, timeout: 15000, maximumAge: 10000
15            },
16        );
17    };
}
```

Code-Snippet 11.5.: JavaScript Funktion - Geolocation-API liefert die Position.

Diese Funktion wird jedes Mal aufgerufen, um die Position zu bestimmen oder noch einmal die Berechtigung anzufragen. Zeile 4 zeigt, dass zuerst geprüft wird, ob die Erlaubnis schon erteilt wurde. Falls nicht wird sie angefragt. Falls dies funktioniert haben sollte, wird in Zeile 5 die Position ausgelesen und, in Zeile 7, abgespeichert. Geolocation ist in der Bibliothek "react-native-geolocation-service" vorhanden.

12. Authentifizierung

Die Umsetzung der Authentifizierung war das wohl komplizierteste Thema in der gesamten Entwicklungsphase. Um die Authentifizierung in der App reibungslos durchzuführen, werden mehrere React Hooks verwendet.

Im Einstiegspunkt der App, also App.js, ist werden mehrere Bibliotheken eingebunden, die in der ganzen App verwendet werden.

```
1 const App = () => (
2   <AuthProvider>
3     <NavigationContainer>
4       <AuthHandler />
5     </NavigationContainer>
6     <FlashMessage position="top" />
7   </AuthProvider>
8 );
9
10 export default App;
```

Code-Snippet 12.1.: React Component - AuthProvider ist die äußerste Komponente.

12.1. AuthContext

AuthContext.js ist für die Speicherung, Änderung und Überprüfung des Json Web Token zuständig. Die Datei besteht aus drei Abschnitten:

12.1.1. state & dispatch

useReducer ist ein React-Hook, welcher in Kombination mit dem Hook useContext eine Alternative zur React-Bibliothek Redux darstellt. Redux wurde entwickelt, um die Speicherung und komplizierte Änderung von Zuständen in React Anwendungen zu vereinfachen. Redux wurde jedoch sehr komplex und benötigte viel Code, um zu funktionieren, daher entwickelten die Ersteller von Redux eine zusätzliche Wrapper-Bibliothek, Redux-Toolkit, um das erstellen von Redux-Stores noch einmal zu vereinfachen [42]. Stores werden in Apps Variablen oder Objekte genannt, die von überall in der App aufrufbar sein sollen und zentrale Informationen zum Zustand speichern.

```

1 const [ state , dispatch ] = useReducer (
2   (prevState , action ) => {
3     switch ( action . type ) {
4       case 'RESTORE_TOKEN' :
5         return {
6           userToken: action . token ,
7           isLoading: false ,
8         };
9       case 'SIGN_OUT' :
10        return {
11           currentUser: null ,
12           userToken: null ,
13           isLoading: false ,
14         };
15        ...
16      }
17    },
18    {
19      isLoading: true ,
20      userToken: null ,
21      currentUser: null ,
22    },
23  );

```

Code-Snippet 12.2.: JavaScript Funktion - Aus dem Hook werden zwei Variablen extrahiert, state und dispatch.

State ist das Objekt, welches wir als zweiten Parameter dem Hook übergeben. Dispatch ist eine Funktion, welche wir aufgerufen werden kann um den State zu verändern.

12. Authentifizierung

```
1 dispatch({ type: 'SIGN_OUT' }) ;
```

Code-Snippet 12.3.: JavaScript Funktion - Die State-Veränderung "SIGN-OUT" wird aufgerufen.

Dispatch wird ein Objekt übergeben, welches im useReducer-Hook action genannt wird. Als erstes wird anhand von action.type in Zeile 3 überprüft, welche Art von Veränderung vorgenommen werden soll. Der Wert der aus der Funktion zurückgegeben wird, wird als neuer State abgespeichert. Es ist auch möglich auf den vorherigen Zustand zuzugreifen, über die Variable prevState.

12. Authentifizierung

12.1.2. authContext

```
1 const authContext = useMemo(            
2   () => ({  
3     restoreToken: async token => {  
4       const res = await fetch(  
5         'https://skate-buddy.josholaus.com/api/users/validate',  
6         {  
7           method: 'POST',  
8           headers: {  
9             Accept: 'application/json',  
10            'Content-Type': 'application/json',  
11            Authorization: `Bearer ${token}`,  
12          },  
13          body: JSON.stringify({ token }),  
14        },  
15      );  
16  
17      if (res.status === 200) {  
18        authContext.decodeToken(token);  
19        dispatch({ type: 'RESTORE_TOKEN', token });  
20      } else {  
21        dispatch({ type: 'SIGN_OUT' });  
22      }  
23    },  
24    signIn: async data => {  
25      ...  
26    },  
27    ...  
28  }) ,  
29  [] ,  
30);
```

Code-Snippet 12.4.: JavaScript Funktion - Die Auth-Funktionen

In authContext werden alle Funktionen erstellt, welche für die Server-Anfragen der Authentifikation wichtig sind. RestoreToken ist die Funktion, die als erstes aufgerufen wird, nachdem die App gestartet wurde. Sie schickt eine POST-Anfrage an den Server (Zeile 4) und übergibt im Content den Token, den es zu überprüfen gilt (Zeile 13). Diese Anfrage wird mit dem await-Statement abgewartet. Nachdem die Antwort eingetroffen ist, wird noch die Information aus dem Token entnommen (Zeile 18) und anschließend abgespeichert, mit dem dispatch-Aufruf in Zeile 19. Sollte der Token nicht gültig oder die Anfrage einfach nicht funktioniert haben, so wird der Benutzer ausgeloggt (Zeile 21).

12. Authentifizierung

12.1.3.AuthProvider

Alle diese Funktionen und Variablen sind enthalten in der Komponente AuthProvider. Sie übernimmt die wichtigste Aufgabe von allen, nämlich das Bereitstellen all dieser Funktionalität an die restliche App.

```
1 const AuthContext = React.createContext();
2 const AuthContextState = React.createContext();
3
4 const useAuthContext = () => useContext(AuthContext);
5 const useAuthContextState = () => useContext(AuthContextState);
6
7 const AuthProvider = ({ children }) => {
8   const [state, dispatch] = useReducer(
9     ...
10   );
11
12   const authContext = useMemo(
13     ...
14   );
15
16   return (
17     <AuthContext.Provider value={authContext}>
18       <AuthContextState.Provider value={state}>
19         {children}
20       </AuthContextState.Provider>
21     </AuthContext.Provider>
22   );
23 };
24
25 export { AuthProvider };
26 export { useAuthContext };
27 export { useAuthContextState };
```

Code-Snippet 12.5.: React Component - Alle Kinder erhalten Zugriff auf authContext und state

AuthProvider umschließt in App.js alle anderen Komponenten, hier wird offensichtlich warum das so sein muss. Zuerst muss aber geklärt werden, was ein Context ist.

In Zeile 3 wird als erstes ein Context erstellt. Dieser erhält den Namen AuthContext und soll die Funktionen aus authContext in der restlichen App verfügbar machen.

In Zeile 19 wird über AuthContext.Provider ein ContextProvider erstellt, der eine Variable über das Property value annimmt.

Wird nun in einer Komponente, die ein Kind von AuthContext ist, die Funktion useContext(AuthContext) aufgerufen, so ist der zurückgegebene Wert gleich der value, die dem Provider übergeben wurde.

Um zu umgehen, dass ich jedes mal, wenn ich eine Funktion brauche, useContext und zusätzlich

12. Authentifizierung

auch noch AuthContext importieren muss, fasse ich diesen Ausdruck zu useAuthContext zusammen (Zeile 6). So ist es möglich, durch das importieren von useAuthContext in einem Kind von AuthContext.Provider auf die gespeicherte Variable zuzugreifen.

In einer üblichen Anwendung werden Daten über Props von Eltern an Kinder weitergegeben. Context macht es möglich direkt für alle Kinder eine Variable zur Verfügung zu stellen, ohne jedem einzeln die Variable zu übergeben.

12.2. AuthHandler

In AuthHandler entscheidet die App, ob ein Benutzer eingeloggt ist oder nicht.

```

1 const Auth = () => {
2   const { restoreToken } = useAuthContext();
3   const state = useAuthContextState();
4
5   useEffect(() => {
6     const checkForValidToken = async () => {
7       let token = null;
8       if (await Keychain.hasInternetCredentials('jwt')) {
9         const creds = await Keychain.getInternetCredentials('jwt');
10        token = creds.password;
11      }
12      restoreToken(token);
13    };
14
15    checkForValidToken();
16  }, [restoreToken]);
17
18  if (state.isLoading) return <SplashScreen />;
19  if (state.userToken == null) return <LoginSignupStack />;
20
21  return <BottomTabsNavigator />;
22};
23
24 export default Auth;

```

Code-Snippet 12.6.: React Component - Ob ein Benutzer eingeloggt ist, hängt von state.userToken ab.

Als erstes werden die Funktion restoreToken und state aus dem Content entnommen. Sobald die App gestartet wird, wird die asynchrone Funktion checkForValidToken aufgerufen. In ihr wird in Zeile 10 und 11 zuerst überprüft, ob im verschlüsselten, internen Speicher ein Eintrag für "jwt" besteht.

Die Komponente Keychain wird von der Bibliothek react-native-keychain bereitgestellt. Sie enthält Funktionen um auf Android und iOS sensible Informationen sicher abzuspeichern. Auf Android wird der Android Keystore verwendet, auf iOS die Keychain Services.

Die Keychain ist in unserer App dafür zuständig, den JWT auch nach dem Beenden der App im internen Speicher festzuhalten und beim Start wieder daraus auszulesen. Sollte ein Eintrag vorhanden sein, wird der Token der Funktion restoreToken aus authContext übergeben, in welcher er überprüft wird.

Beim Start der App wurde state.isLoading als true und state.userToken als null definiert. So lange isLoading auf true ist, wird der Ladebildschirm SplashScreen angezeigt. Sobald der Token überprüft wurde, wird isLoading auf false gesetzt und es entscheidet sich ob der User eingeloggt

12. Authentifizierung

wird, oder nicht.

Ist nach der Überprüfung die Variable userToken immer noch null, so wird der LoginStack angezeigt. Sollte der Token gültig sein, so wird er in userToken abgespeichert, was die If-Abfrage in Zeile 27 false macht. Somit bleibt nur noch das Statement in Zeile 31 übrig, der User ist eingeloggt.

13. Navigation

Für die Navigation innerhalb der App wurde auf intuitives und simples Design gesetzt, es sollte von jedem sofort verstanden werden. Natürlich ist das Selbst-Implementieren dieser Funktionalität keine Option, daher wurde die Bibliothek React Navigation verwendet. Sie verknüpft die einzelnen Screens der App miteinander und stellt das Gegenstück zu React Router für React im Webbrower dar.

Damit die Navigation funktionieren kann, muss die gesamte App ein Kind von NavigationContainer sein, einer Komponente, die von React Navigation importiert wird.

```
1 import { NavigationContainer } from '@react-navigation/native';
2
3 const App = () => (
4   <AuthProvider>
5     <NavigationContainer>
6       <AuthHandler />
7     </NavigationContainer>
8     <FlashMessage position="top" />
9   </AuthProvider>
10 ) ;
```

Code-Snippet 13.1.: JavaScript Funktion - In NavigationContainer können Navigationen erstellt werden.

Die Navigation innerhalb der App wurde auf zwei Arten umgesetzt:

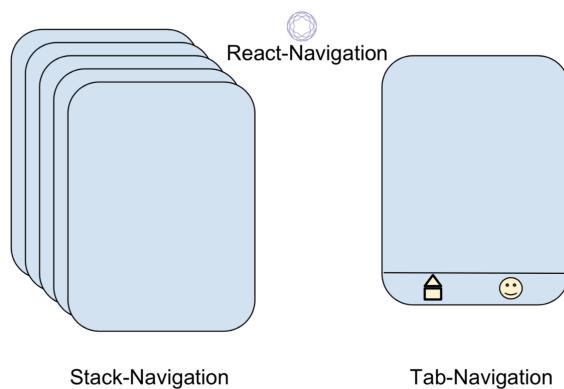


Abbildung 13.1.: Stack Navigator und Tab-Navigator

13. Navigation

13.1. Tab-Navigation

Andere mit React Native gebauten Apps, wie Facebook oder Instagram, verwenden eine sogenannte "Bottom-Tabs"-Navigation, also eine Tab-Basierte Navigation auf der Unterseite des Bildschirms, um ihre Hauptfunktionen dem Benutzer zu präsentieren.

Um die App simpel zu halten ist dies auch unsere präferierte Navigations-Methode, um die einzelnen Bereiche der Anwendung miteinander zu verbinden.

```
1 import MapScreen from '../screens/MapScreen';
2 import ProfileScreen from '../screens/ProfileScreen';
3 import SkateparksStack from './SkateparksStack';
4 import Colors from '../styles/Colors';
5
6 const Tab = createBottomTabNavigator();
7
8 const BottomTabsNavigator = () => {
9   const tabBarIcons = (route, focused, color) => {
10     ...
11   };
12   return (
13     <Tab.Navigator
14       backBehavior="initialRoute"
15       initialRouteName="Skateparks"
16       screenOptions={({ route }) => ({
17         tabBarIcon: ({ focused, color }) => tabBarIcons(route,
18           focused, color),
19         tabBarActiveTintColor: Colors.primary,
20         tabBarInactiveTintColor: Colors.gray2,
21         tabBarActiveBackgroundColor: Colors.primarySoft,
22         tabBarShowLabel: false,
23         headerShown: false,
24       })}
25     >
26       <Tab.Screen name="Skateparks" component={SkateparksStack} />
27       <Tab.Screen name="Map" component={MapScreen} />
28       <Tab.Screen name="Profile" component={ProfileScreen} />
29     </Tab.Navigator>
30   );
31 }
32
33 export default BottomTabsNavigator;
```

Code-Snippet 13.2.: React Component - Bottom-Tab-Navigation

In Zeile 14 wird aus dem Objekt Tab zuerst der Navigator und danach, als Kinder, die einzelnen Screens eingefügt. Als Screens können hierbei weitere andere Navigatoren verwendet werden, um eine Verschachtelung zu erzeugen.

13. Navigation

13.1.1. MapScreen

Die Umsetzung der Kartenfunktion, um alle Skateparks anzuzeigen, war das zweit-komplizierteste Feature in der App. Die Bibliothek "react-native-maps" ist hierbei die Basis dieser Funktion.

Zuerst werden mit dem Hook `useFetch` alle Skateparks von der API abgefragt, solange die Variable `skateparks` null ist, wird nur ein Ladesymbol angezeigt. Anschließend wird die Map erzeugt, wie auch ein Knopf, mit dem man die Position der Karte auf Innsbruck zentrieren kann.

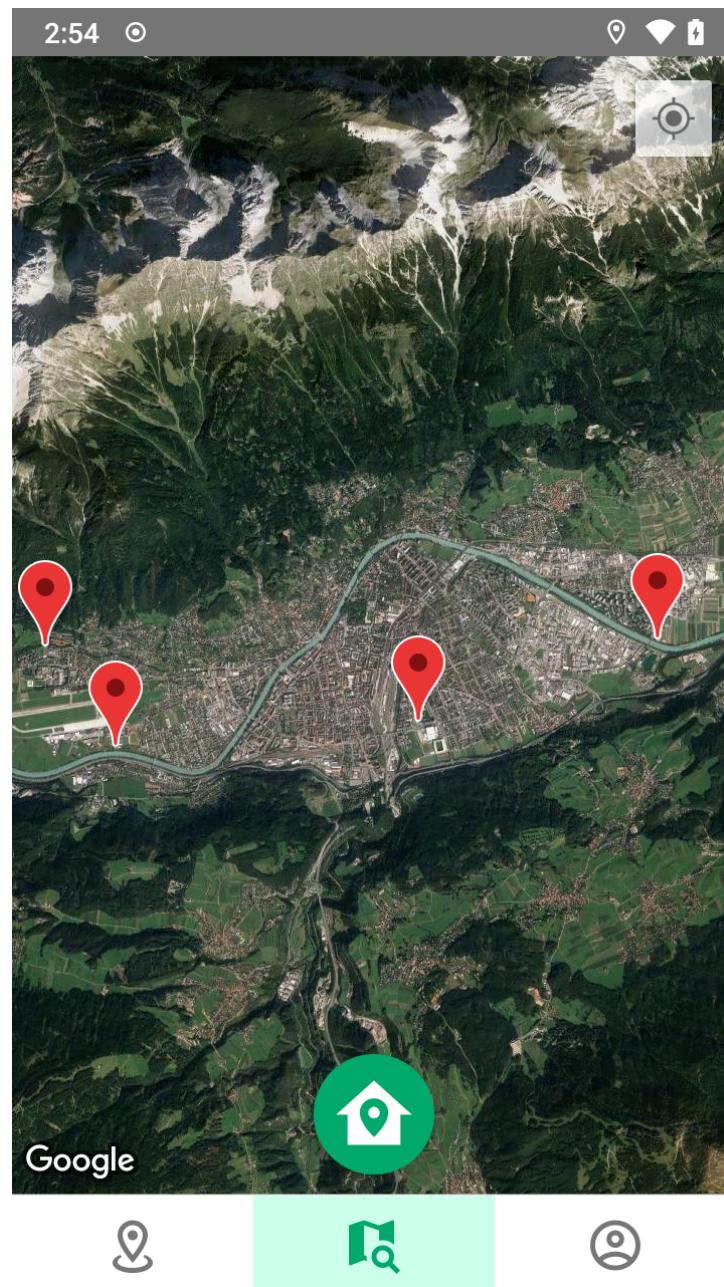


Abbildung 13.2.: Alle Skateparks werden an der korrekten Stelle angezeigt

13. Navigation

```
1 const MapScreen = ({ navigation }) => {
2   const {
3     data: skateparks,
4     isLoading,
5     error,
6     reloadData,
7   } = useFetch('https://skate-buddy.josholaus.com/api/skateparks');
8   const mapRef = useRef(null);
9
10  return (
11    <View style={styles.container}>
12      {isLoading && <LoadingCircle />}
13      {error && <Error error={error} refresh={reloadData} />}
14      {skateparks && (
15        <>
16          <Map
17            mapRef={mapRef}
18            skateparks={skateparks}
19            navigation={navigation}
20          />
21          <CircleButton
22            onPress={() => {
23              mapRef.current.animateCamera({
24                center: {
25                  latitude: 47.27,
26                  longitude: 11.4,
27                },
28                altitude: 1000,
29                pitch: 0,
30                heading: 0,
31                zoom: 12,
32              });
33            }}
34          />
35        </>
36      )}
37    </View>
38  );
39};
40 export default MapScreen;
```

Code-Snippet 13.3.: React Component - Der Karten-Tab

13. Navigation

13.1.1.1. Map

```
1 const Map = ({ skateparks , mapRef , navigation }) => {
2   return (
3     <MapView
4       // props
5       showsUserLocation
6       provider={PROVIDER_GOOGLE}
7       style={mapStyles.map}
8       ref={mapRef}
9       initialCamera={camera}
10      showsPointsOfInterest={false}
11      showsCompass={false}
12      showsIndoors={false}
13      minZoomLevel={8}
14      rotateEnabled={false}
15      pitchEnabled={false}
16      mapType="satellite"
17    >
18    <SkateparkMarkers
19      skateparks={skateparks}
20      mapRef={mapRef}
21      navigation={navigation}
22    />
23    </MapView>
24  );
25};
26 export default Map;
```

Code-Snippet 13.4.: React Component - Die MapView-Komponente aus react-native-maps

In der Komponente Map wird einfach eine MapView aufgerufen, welche die Google Maps Ansicht beinhaltet. Die Parks werden anschließend wieder an die nächste Komponente weitergegeben, wo sie in die Map-Marker umgewandelt werden. Drückt man auf einen Marker, so wird der Park zentriert und ein Textfeld wird angezeigt. Wird dies angeklickt, so wird man zu den Details vom jeweiligen Park weitergeleitet.

13. Navigation

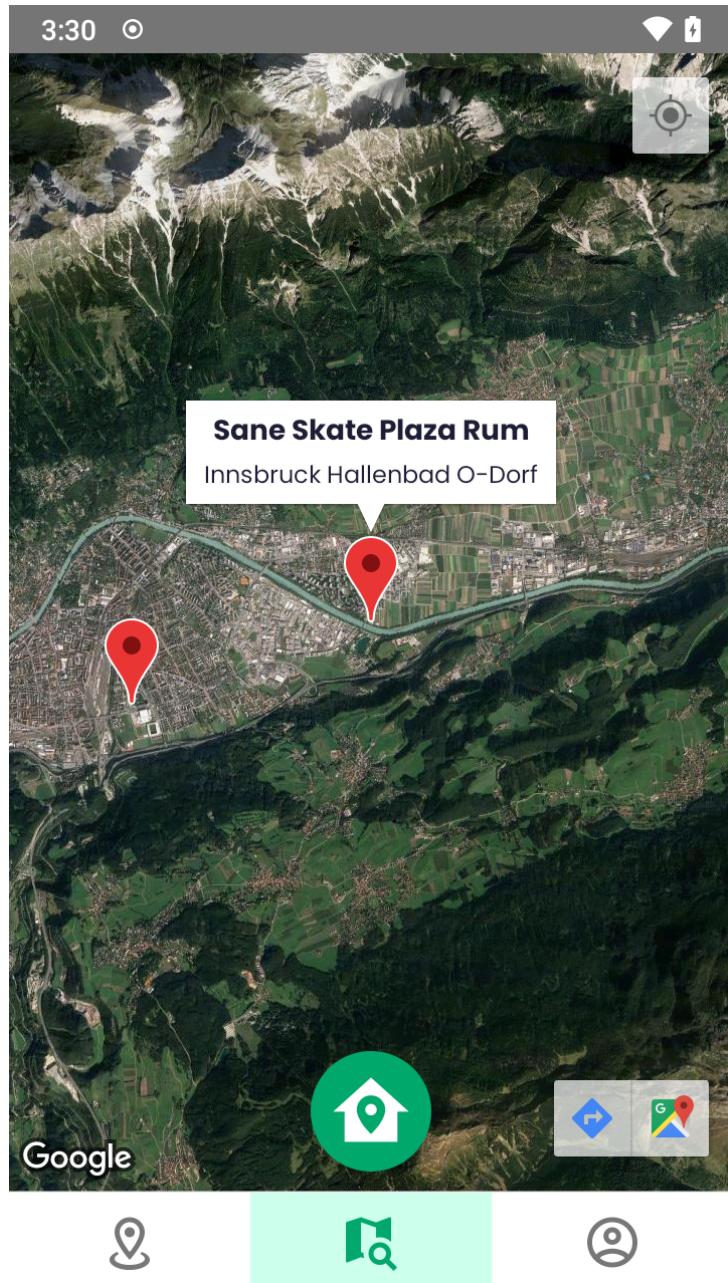


Abbildung 13.3.: Ein ausgewählter Marker

13. Navigation

13.1.2. ProfileScreen

In der Profilansicht werden lediglich einige Informationen zum User, ein Standard-Profilbild und ein Knopf, zum Ausloggen, angezeigt.

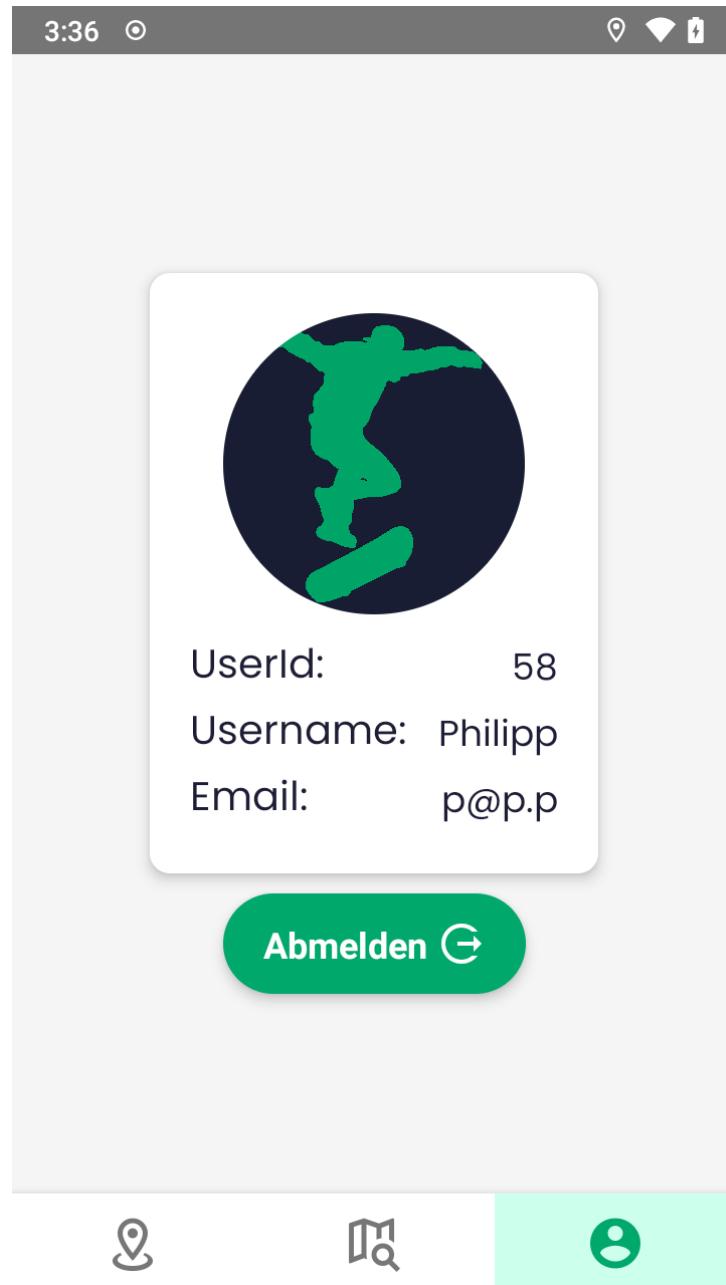


Abbildung 13.4.: Die Profil-Ansicht

13. Navigation

13.2. Stack-Navigation

Ein Stack-Navigator legt einen Screen als Basis-Screen fest und legt jeden neuen Screen darüber. Jeder Screen in einem Stack-Navigator erhält automatisch das `navigation` Objekt als Property. Mit ihm können Funktionen aufgerufen werden, mit der die Navigation durchgeführt wird. Mit dem Befehl `push` wird ein neuer Screen auf den Stack gelegt, mit der Funktion `pop` wird das oberste Element entfernt.

13.2.1. SkateparksStack

13.2.1.1. SkateparksList

Auf der ersten Seite sollte man sofort einen Überblick über alle Skateparks erhalten, die wir in unserer Datenbank gespeichert haben.

13. Navigation

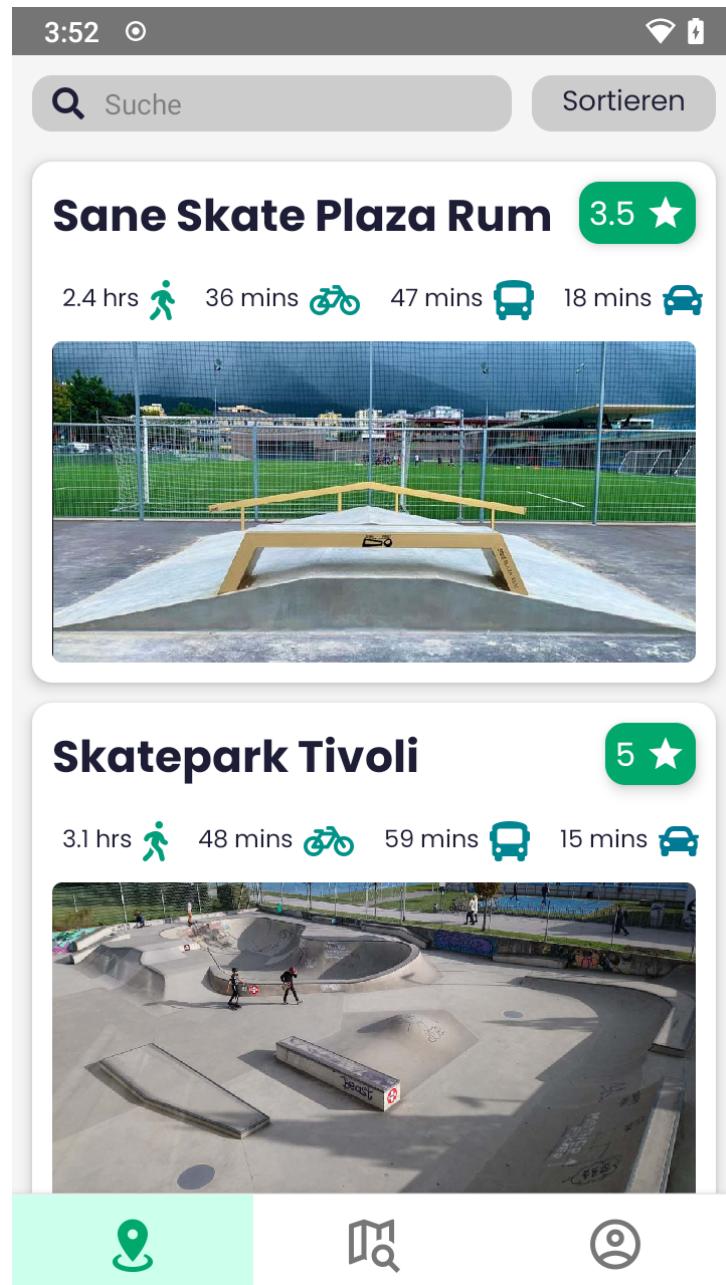


Abbildung 13.5.: SkateparksList Screen

Wenn der Benutzer auf einen Skatepark drückt, wird bei der Navigation zu SkateparkDetails der Skatepark übergeben, um die Informationen anzuzeigen.

Am oberen Rand des Bildschirms hat der Benutzer die Möglichkeit, nach einem bestimmten Skateparknamen zu suchen und nach der Reisezeit zu sortieren, aufsteigend und absteigend.

13. Navigation

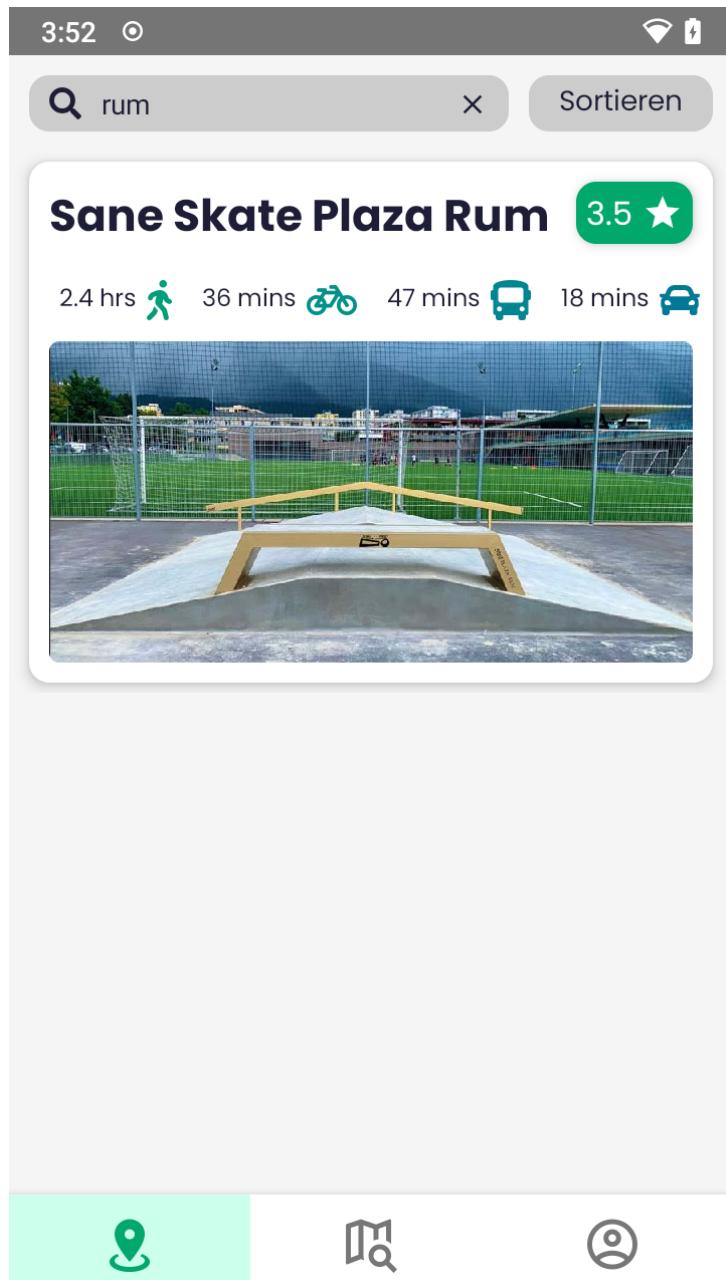


Abbildung 13.6.: Suchfunktion Demonstration

13. Navigation

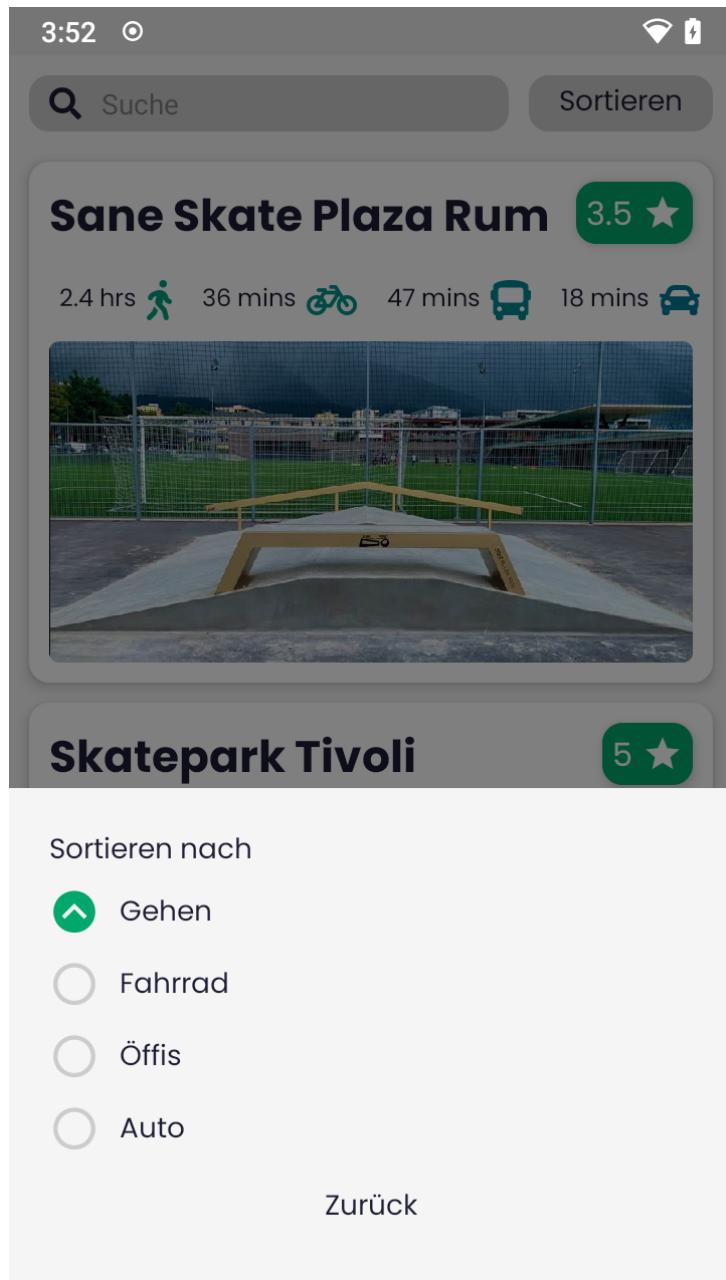


Abbildung 13.7.: Sortieren nach Zeit

13. Navigation

13.2.1.2. SkateparkDetails

Als erstes werden die wichtigsten Informationen zum Standort des Parks angezeigt. Darunter gibt es eine kleine Diashow mit diversen Bildern des Parks.

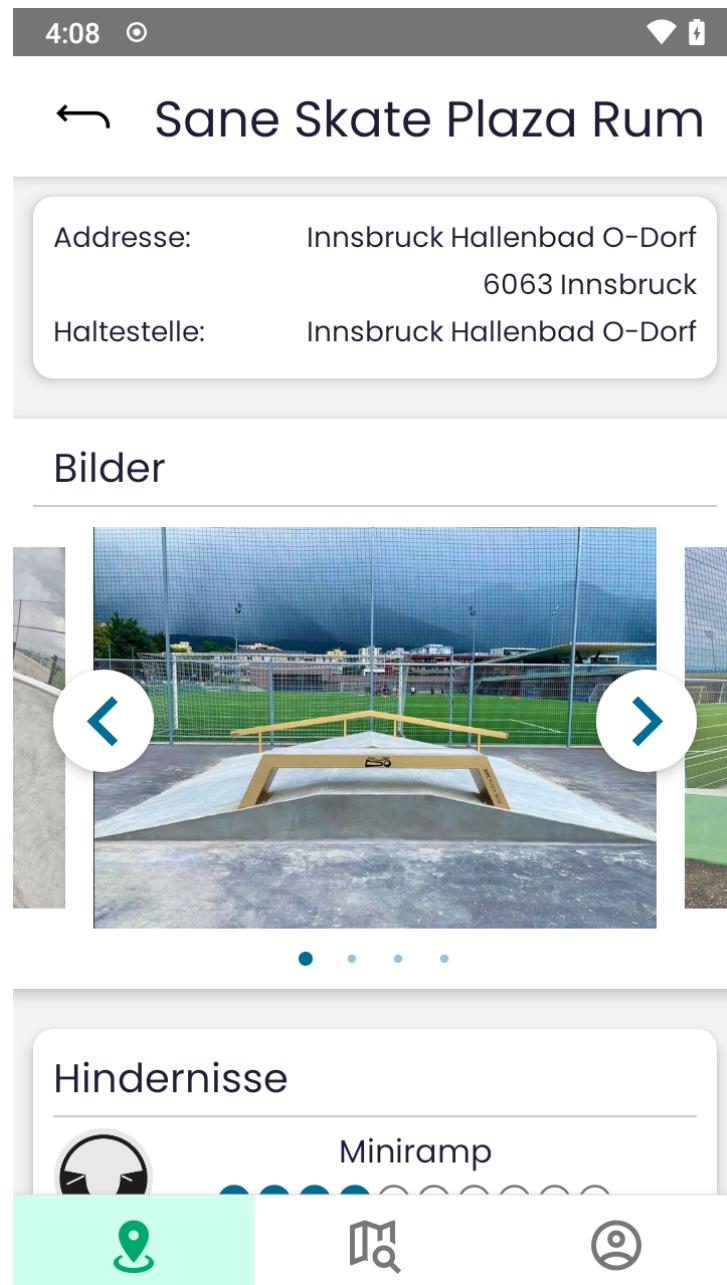


Abbildung 13.8.: Adresse und nächste Haltestelle werden als erstes gezeigt

13. Navigation

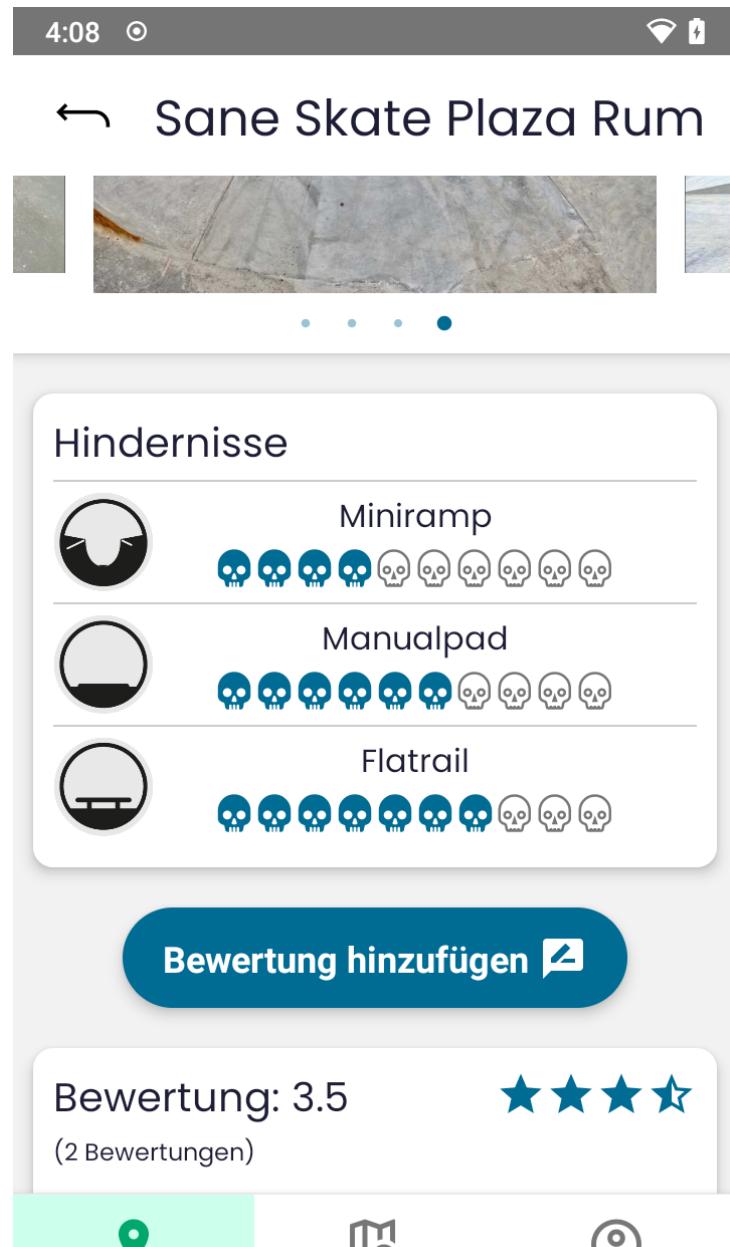


Abbildung 13.9.: Zu jedem Park werden auch Hindernisse mit Schwierigkeitseinschätzungen abgespeichert

Um eine Bewertung zu hinterlassen, drückt der Benutzer auf den Knopf "Bewertung hinzufügen" und füllt das angezeigte Formular aus. Anschließend werden die Daten an das Backend geschickt und abgespeichert.

13. Navigation

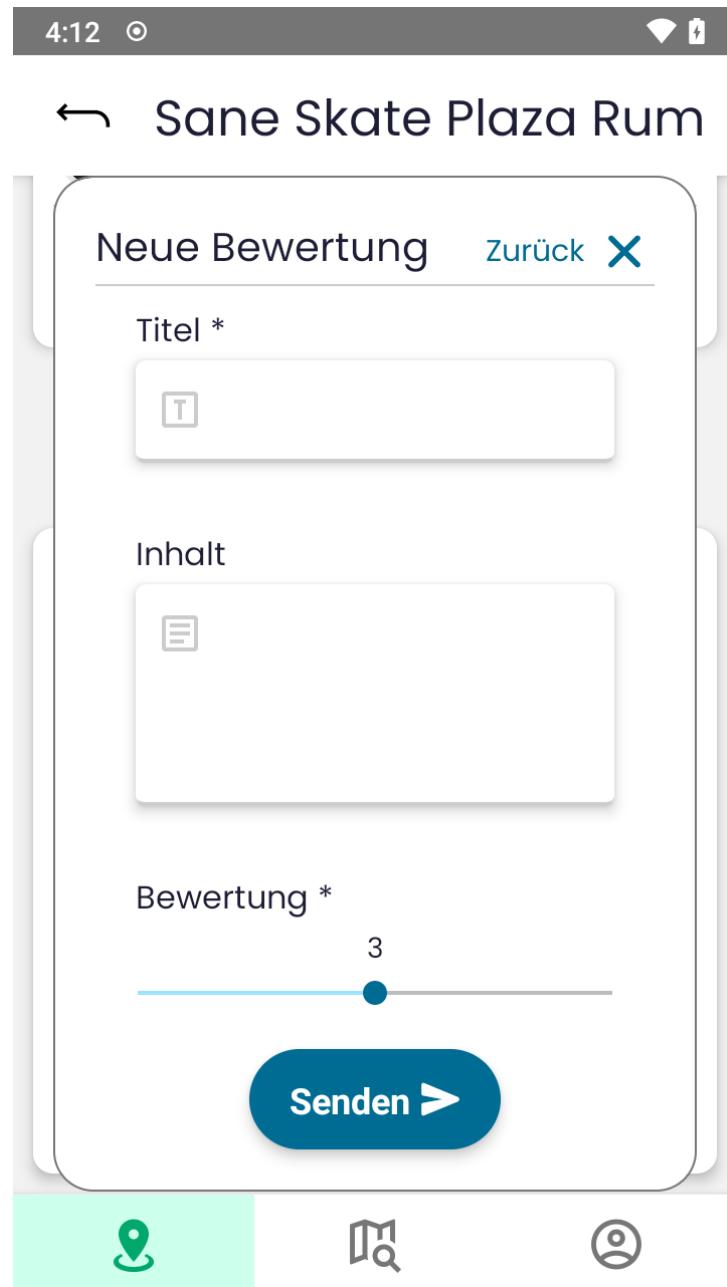


Abbildung 13.10.: Daten eintragen und absenden.

Am Ende wird noch die durchschnittliche Bewertung, die Anzahl der Bewertungen und alle Bewertungen selbst angezeigt.

13. Navigation

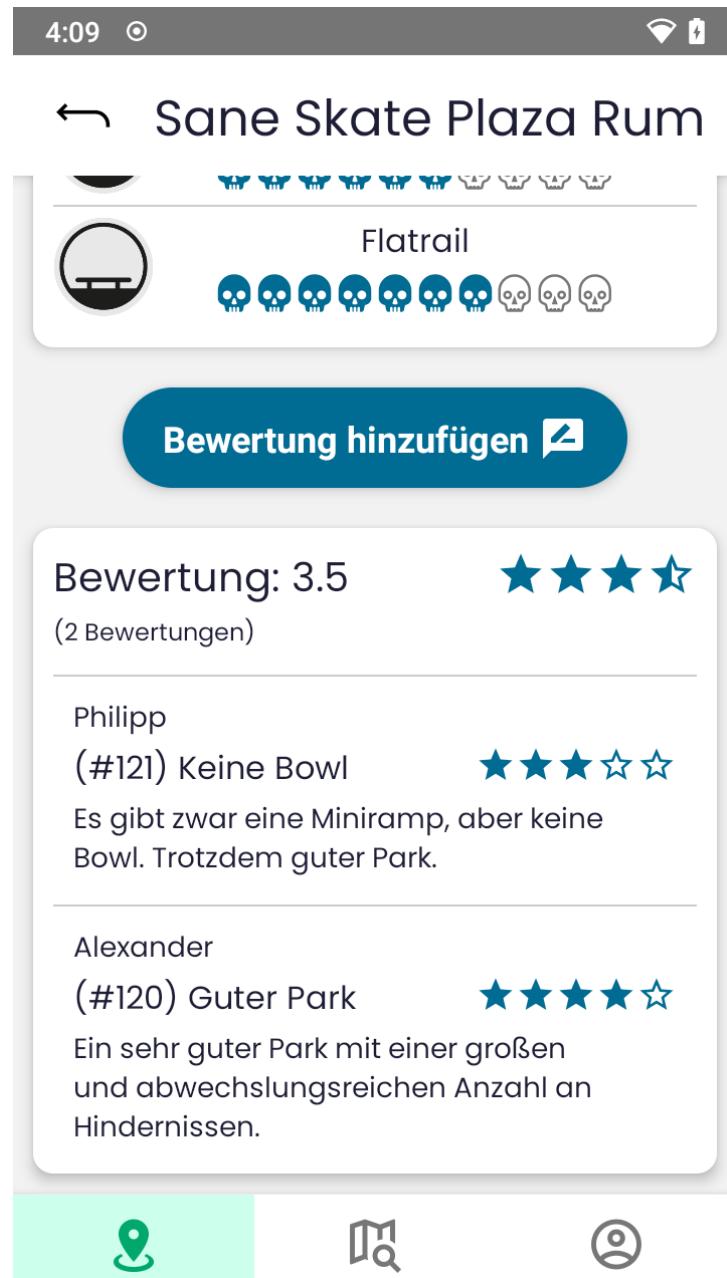


Abbildung 13.11.: Alle Bewertungen

13.2.2. LoginSignupStack

Der LoginSignupStack besteht aus drei Screens:

13.2.2.1. LoginScreen

Im LoginScreen wird der Benutzer dazu aufgefordert, seine Anmeldedaten einzugeben.

13. Navigation

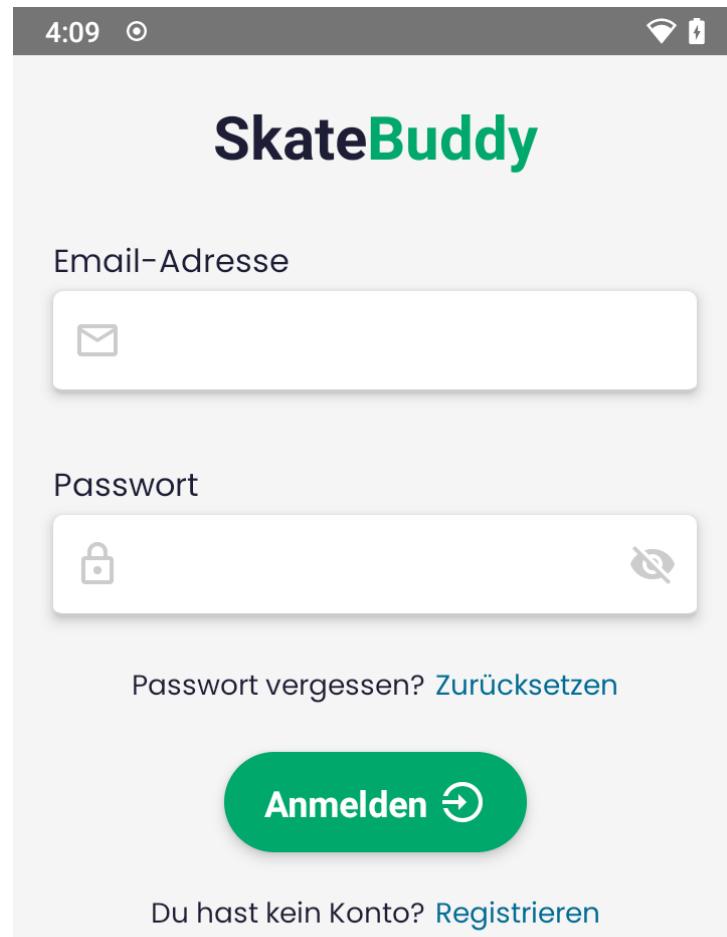


Abbildung 13.12.: Daten eintragen und absenden.

13.2.2.2. SignupScreen

Sollte er noch keinen Account haben, ist es ihm möglich, einen neuen Account zu erstellen über den SignupScreen.

13. Navigation

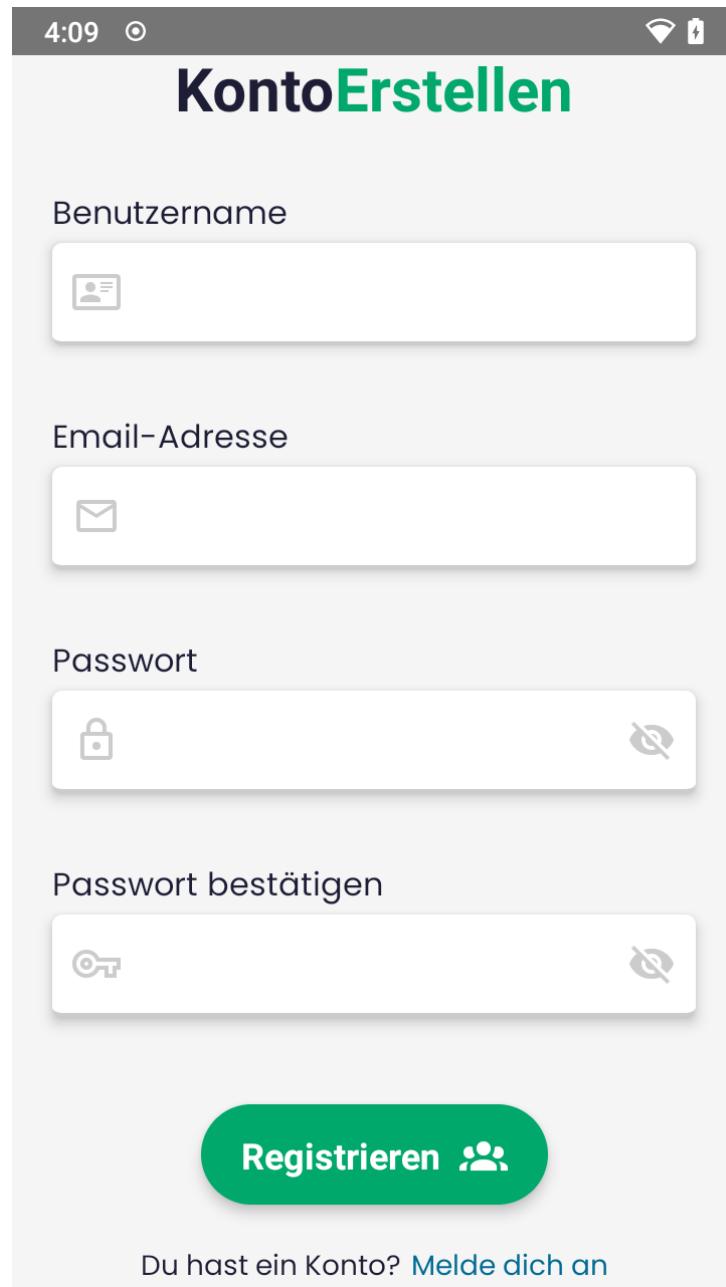


Abbildung 13.13.: Daten eintragen und absenden.

13.2.2.3. ForgotPassword

Wenn es dem Benutzer passieren sollte, dass er sein Passwort vergisst und keinen Zugriff auf seinen Account mehr hat, so kann er über diesen Bildschirm einen Link anfordern, mit dem er sein Passwort zurücksetzen kann. Dieser Link wird per E-Mail versendet.

Diese Funktion ist noch nicht in unserer App vorhanden, bis jetzt wird einfach überprüft, ob diese E-Mail-Adresse existiert.

13. Navigation

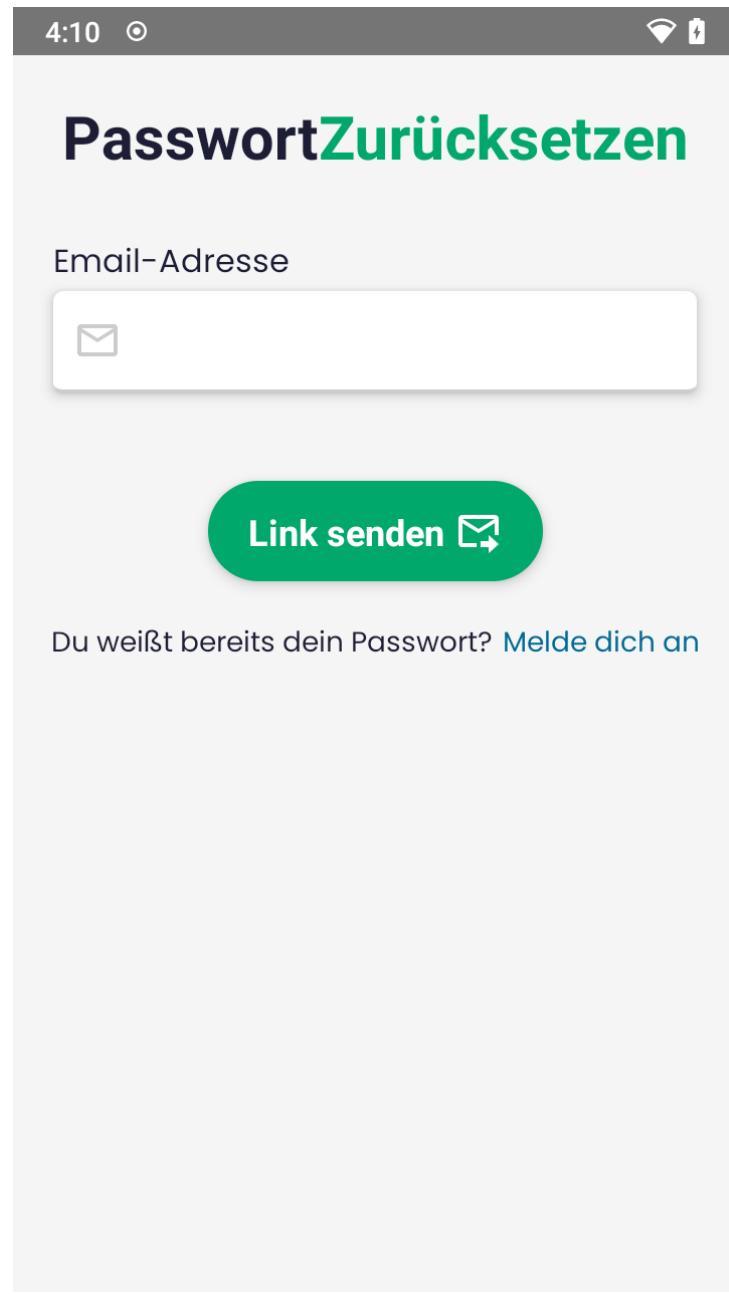


Abbildung 13.14.: Daten eintragen und absenden.

Teil V.

Backend

14. Allgemeines

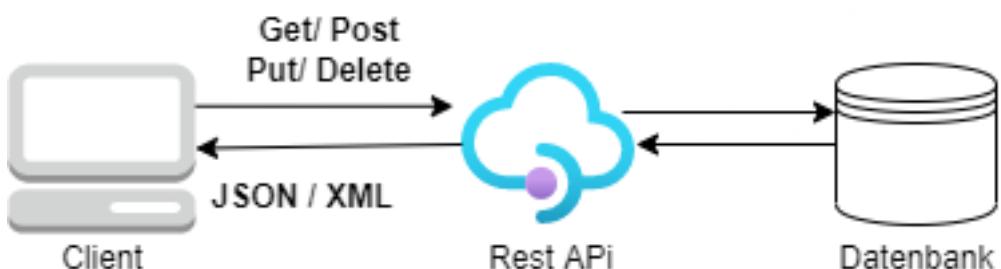
Wie bereits im Kapitel Server erwähnt 5.1.1.5. Hat der **Backend-Server** die Aufgabe, Daten aus der Datenbank an die User bzw. Clients zu senden. Dies erledigt er durch Abruf bestimmter Routes. Für die Entwicklung des Backend-Servers haben wir uns für die Programmiersprache *Javascript* entschieden 3.4.

14.1. Datenbankverbindung

Beim Start der Anwendung wird mit Hilfe der *mySQL* Library eine Verbindung zu mySQL Servern aufgebaut, welche sich in einem separaten Docker Container läuft. Die Anmeldedaten sowie die Port Nummer sind in den Environment Variablen gespeichert.

14.2. Rest-API

Rest steht für REpresentational State Transfer, **API** für Application Programming Interface. Die Rest API hat die Aufgabe den Client bzw. die App mithilfe von sogenannten **Rest-Routes** zugriff auf die Datenbank zu geben. Im Prinzip kann jeder Computer mit Internet Zugang auf die API zugreifen, insofern sie die vorgefertigten **HTTP-Anfragen** an den Server senden. Wie genau diese Anfragen aussehen ist im Kapitel **Rest-Routes** beschrieben 14.2.2.1.



14.2.1. HTTP / HTTPS

HTTP(s) steht für HyperTextTransferProtocol(secure). Dieses Protokoll regelt wie eine Seite vom Server zum Client übertragen wird. Es wird jedoch ebenfalls zur Steuerung von Rest-API's verwendet. Der Client kann an den Server folgende **HTTP/HTTPS** Anfragen senden:

14. Allgemeines

HTTP-Methode (Befehl)	Definition
Get	Fordert eine Ressource vom Server an ohne sie zu
POST	Erstellt eine neue Ressource und sendet diese an den Server
PUT	Legt die angegebene Ressource auf dem Server an oder modifiziert eine bestehende
PATCH	Modifiziert einen Teil der angegebenen Ressource
DELETE	Löscht eine Angegebene Ressource

14.2.2. Authorization

Damit sensible Daten wie zum Beispiel die User Daten nicht für alle PC's frei zugänglich sind werden bestimmte *Routes* mit einer *Bearer Authorization* gesichert. Um die Routes benutzen zu können muss im Header der HTTP Anfrage ein Authorization Eintrag mitgesendet werden. Falls keiner mitgesendet wird sendet die API eine Message mit den Inhalt **Unauthorized**.

Nur Benutzer die einen sogenannten *Bearer-Token* besitzen haben also Zugriff auf die Ressourcen.

Einen solchen Token erhält man entweder wenn man sich registriert siehe (api/register) oder wenn sich anmeldet siehe (api/login).

Wie so ein Bearer Token aussieht sehen sie hier:

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VySWQiOjMyLCJuYW1lIjoiQSIsImVtYWIsljoiQUBBIiwicHJvZmlsZXByZ3R1cmVJZCI6bnVsbCwiaWF0IjoxNjQ1Mjk1Nzk0LCJleHAiOjE2TY5NTk3OTR9.k1p8nju8ra6Rt16nZTTCmrkrhK3H8MygbwGYHtio-Sg
```

Als Bearer Token haben wir uns für den *Json Web Token (JWT)* entschieden.

14.2.2.1. Json Web Token

Der *JWT* wird in der Regel für Authentifizierungsvorgängen verwendet. Es ist ebenfalls möglich, in ihm Informationen zu speichern. Die Informationen können verschlüsselt werden und somit sichergestellt werden, wer der Absender ist und ob dieser die benötigten Zugriffsrechte hat. Ein solcher Token besteht aus drei Teilen.

Header Der Header besteht aus zwei Teilen und liefert Informationen über den Token. Welcher Typ und welcher Verschlüsselungsalgorithmus angewendet wurde.

Payload In der Payload wird die tatsächliche Information des Tokens gespeichert. Die Informationen werden als **Key/Value Paare** bereitgestellt.

Signature Die Signatur des Schlüssels wird mithilfe der *Base-64 Kodierung* der Headers der Payload und der angegebenen Signatur-/Verschlüsselungsmethode erstellt. Damit die Signatur gültig ist wird ebenfalls noch ein geheimer Schlüssel benötigt dem nur der Ursprungsquelle bekannt ist. Somit wird sichergestellt das der Token nicht verändert wurde.

14. Allgemeines



<https://medium.com/swlh/understand-the-concept-of-jwt-json-web-tokens-4ee18682a5>

14. Allgemeines

```
1 router.use('/users*', async (req, res, next) => {
2   const auth = req.headers['authorization'];
3
4   if (!auth) {
5     res.sendStatus(401);
6     return;
7   }
8   const token = auth.split(' ');
9   try {
10     jwt.verify(token[1], process.env.JWT_HASH_SECRET);
11     next();
12   } catch (e) {
13     res.sendStatus(401);
14   }
15 }) ;
```

Code-Snippet 14.1.: Code-Snippet-Authorization

In der ersten Zeile sieht man das diese Route für alle http Anfragen die mit `/api/users` beginnt gilt. In der Konstante `auth` wird der Header ausgelesen. Falls sich im Header nichts befindet sendet der Server einen Error 401 **Unauthorized**. Falls es einen Token findet wird er gesplited und in eine Konstante namens `token` gespeichert. Nun wird in der 10 Zeile der Token mithilfe der Methode `jwt.verify(token[1], Hash SECRET)` verifiziert. Falls dies klappt wird die Methode `next()` aufgerufen die weiter `/api/users` anfragen durchlässt.

[43]

15. Rest-Routes

Im folgendem Kapitel werden alle **Rest-Routes** der API dokumentiert und erklärt. Die API ist nicht für den öffentlichen Gebrauch konzipiert und wurde ausschließlich für die App und die Website **Skate-Buddy** erstellt.

15.1. Route-Routes

15.1.0.1. GET /users

1 /users

Zweck der Route:

Gibt eine JSON liste zurück mit allen vorhandenen Usern.

Authorization:

Bearer

Zusätzliche Parameter **Body:** Keinen

Body: Keinen

15. Rest-Routes

15.1.0.2. Response

Content-Type: JSON

```
1  [
2  {
3      "userId": 54,
4      "name": "ww",
5      "passwordhash": "$2b$10$6TnxbdU8GnROQK5oLAmEv.
6          RcrIF9MCfC7Jlyv653L45FizJMGuz3u",
7      "email": "w@w.w",
8      "profilepictureId": null,
9      "admin": 0
10 },
11 {
12     "userId": 57,
13     "name": "Alexander",
14     "passwordhash": "$2b$10$rYPBy6ijGCb.Z/ZevSyJDuk4UXZnPnnMKW4Y1X/
15         zH1OUYNx93zUq",
16     "email": "abertoni@tsn.at",
17     "profilepictureId": null,
18     "admin": 1
19 },
20 {
21     "userId": 58,
22     "name": "Philipp",
23     "passwordhash": "$2b$10$IItk58v.13
24         txEcsTqd2cIuI7T3v9XJSt6BK5Rs87cCbutBrijTwqi",
25     "email": "p@p.p",
26     "profilepictureId": null,
27     "admin": 0
28 }
29 ]
30 ]
```

15. Rest-Routes

15.1.0.3. GET /users/:id

```
1 /users/54
```

Zweck der Route

Gibt einen User mit einer bestimmten ID zurück.

Zusätzliche Parameter id: int (benötigt)

→ Die eindeutige ID des gewünschten Users

```
1 GET /users/54
```

Body: Keinen

Authorization:

Bearer

Response Content-Type: JSON

```
1 [
2 {
3     "userId": 54,
4     "name": "ww",
5     "passwordhash": "$2b$10$6TnxbdU8GnROQK5oL
6                     AmEv.RcrlF9MCfC7Jlyv653I45FizJMGuz3u",
7     "email": "w@w.w",
8     "profilepictureId": null,
9     "admin": 0
10    },
11 ]
```

15. Rest-Routes

15.1.0.4. /users/decode

```
1   /users/decode
```

Zweck der Route:

Im Body wird ein JSON Web token mitgesendet. Als Antwort wird die Payload ausgeben.

Authorization-Typ:

Bearer

Zusätzliche Parameter Keine

Body:

```
1
2   "token": "eyJhbGciOiJIUzI1NJ.LCJleHAiOY5NTk3OTR9.k1p8njURt1GYHtio-Sg"
```

15.1.0.5. Response:

Content-Type: Json

```
1   [
2     "userId": 32,
3     "name": "A",
4     "email": "A@A",
5     "profilepictureId": null,
6     "iat": 1645295794,
7     "exp": 1656959794
8   ]
```

15. Rest-Routes

15.1.0.6. GET /users/validate

```
1 /users/validate
```

Zweck der Route:

Im Body wird ein JWT-Token gesendet und der Server gibt zurück ob der Token gültig ist oder nicht.

Authorization:

Bearer

Zusätzliche Parameter ID

Body: Content-Type: JSON

```
1 {
2     "token": "eyJhbGciOiJIUzI1NJ.LCJleHAiOY5NTk3OTR9.k1p8njURt1GYHtio-Sg"
3 }
```

15.1.0.7. Response)

Content-Type: JSON

```
1 {
2     "success": true,
3     "message": "Token Valid"
4 }
```

15. Rest-Routes

15.1.0.8. DEL /users/

```
1 /users/1
```

Zweck der Route:

Ein User mit einer bestimmten ID wird aus der Datenbank gelöscht.

Authorization:

Bearer

Zusätzliche Parameter id: int (benötigt) → Die eindeutige ID des gewünschten Users

Body: Keinen

15.1.0.9. Response)

Content-Type: JSON

```
1 {
2     "success": true,
3     "message": "Successfully deleted"
4 }
```

15. Rest-Routes

15.1.0.10. PUT /users/:id

```
1 /users/1
```

Zweck der Route:

Ein User mit einer bestimmten ID wird aus der Datenbank kann nach belieben geändert werden.

Authorization:

Bearer

Zusätzliche Parameter id: int (benötigt) → Die eindeutige ID des gewünschten Users

Body:

Content-Type: JSON

```
1 {
2     {
3         "column": "name",
4         "newValue": "hans"
5     }
6 }
7 }
```

15.1.0.11. Response)

Content-Type: JSON

```
1 {
2     {
3         "success": true,
4         "message": "Successfully updated"
5     }
6 }
```

15. Rest-Routes

15.1.0.12. POST /register

```
1   /register
```

Zweck der Route:

Mit dieser Route kann man sich registrieren. Der Server sendet dann eine Nachricht mit einem Erfolg oder nicht Erfolg. Bei dieser Route erhält der User einen gültigen Bearer Token.

Authorization:

Keine

Zusätzliche Parameter

Keine

Body: Content-Type: JSON

```
1   {
2     {
3       "name": "Test",
4       "password": "Test",
5       "email": "test@test.com"
6     }
7 }
```

15.1.0.13. Response)

Content-Type: JSON

```
1   {
2     {
3       "success": true,
4       "token": "eyJhbGciOiJIUzI1Nj.LCJleHAiOY5NTk3OTR9.k1p8njURT1GYHtio-Sg"
5     }
6 }
```

15. Rest-Routes

15.1.0.14. POST /login

```
1   /login
```

Zweck der Route:

Mit dieser Route kann man sich einloggen. Der Server sendet dann eine Nachricht mit einem Erfolg oder nicht Erfolg. Bei dieser Route erhält der User einen gültigen Bearer Token.

Authorization:

Keine

Zusätzliche Parameter

Keine

Body: Content-Type: JSON

```
1   {
2     {
3       "name": "Test",
4       "password": "Test",
5       "email": "test@test.com"
6     }
7 }
```

15.1.0.15. Response)

Content-Type: JSON

```
1   {
2     {
3       "success": true,
4       "token": "eyJhbGciOiJIUzI1Nj.LCJleHAiOY5NTk3OTR9.k1p8njURT1GYHtio-Sg"
5     }
6 }
```

15. Rest-Routes

15.1.0.16. GET /reviews

```
1   /reviews
```

Zweck der Route:

Gibt eine JSON liste zurück mit allen vorhandenen Reviews.

Authorization:

Keine

Zusätzliche Parameter Keine

Body: Keinen

15.1.0.17. Response)

Content-Type: JSON

```
1   [
2     {
3       "reviewId": 122,
4       "skateparkId": 2,
5       "userId": 58,
6       "rating": 5,
7       "title": "Grosse Bowl",
8       "content": "Einfach nur perfekt.",
9       "username": "Philipp"
10      },
11      {
12        "reviewId": 121,
13        "skateparkId": 1,
14        "userId": 58,
15        "rating": 3,
16        "title": "Keine Bowl",
17        "content": "Es gibt zwar eine Miniramp, aber keine Bowl. Trotzdem guter
18          Park.",
19        "username": "Philipp"
20      },
21      {
22        "reviewId": 120,
23        "skateparkId": 1,
24        "userId": 57,
25        "rating": 4,
26        "title": "Guter Park",
27        "content": "Ein sehr guter Park mit einer grossen und
28          abwechslungsreichen Anzahl an Hindernissen.",
29        "username": "Alexander"
30      }
]
```

15. Rest-Routes

15.1.0.18. GET /reviews/:id

```
1   /reviews/2
```

Zweck der Route

Gibt einen Review mit zu einem bestimmten Skatepark zurück.

Zusätzliche Parameter id: int (benötigt)

→ Die eindeutige ID des gewünschten Skateparks

Body:

Keinen

Authorization:

Keinen

Response Content-Type: JSON

```
1   [
2     {
3       "reviewId": 122,
4       "skateparkId": 2,
5       "userId": 58,
6       "rating": 5,
7       "title": "Grosse Bowl",
8       "content": "Einfach nur perfekt.",
9       "username": "Philipp"
10      }
11  ]
```

15. Rest-Routes

15.1.0.19. Post /reviews

```
1   /reviews/2
```

Zweck der Route

Mit dieser Route ist es möglich eine neue Review zu einem bestimmten Skatepark hinzuzufügen.

Zusätzliche Parameter `id: int` (benötigt)

→ Die eindeutige ID des gewünschten Skateparks

Body:

Content-Type: JSON

```
1   {
2     "parkid": "2",
3     "userid": "10",
4     "rating": "5",
5     "title": "Cool",
6     "content": "Nice"
7 }
```

Authorization:

Keinen

Response Content-Type: JSON

```
1 [
2   {
3     "success": true,
4     "message": "Successfully inserted"
5   }
6 ]
```

15. Rest-Routes

15.1.0.20. DEL /reviews/

```
1   /users/1
```

Zweck der Route:

Eine Review mit einer bestimmten ID wird aus der Datenbank gelöscht.

Authorization:

Bearer

Zusätzliche Parameter id: int (benötigt) → Die eindeutige ID der gewünschten Review.

Body: Keinen

15.1.0.21. Response)

Content-Type: JSON

```
1   {
2     "success": true,
3     "message": "Successfully deleted"
4 }
```

15. Rest-Routes

15.1.0.22. PUT /reviews/:id

```
1   /reviews/1
```

Zweck der Route:

Eine Review mit einer bestimmten ID kann nach belieben geändert werden.

Authorization:

Keine

Zusätzliche Parameter id: int (benötigt) → Die eindeutige ID der gewünschten Review.

Body:

Content-Type: JSON

```
1   {
2     {
3       "Title": "Alter Titel",
4       "newValue": "Neuer Titel"
5     }
6   }
```

15.1.0.23. Response)

Content-Type: JSON

```
1   {
2     {
3       "success": true,
4       "message": "Successfully updated"
5     }
6   }
```

15. Rest-Routes

15.1.0.24. GET /skateparks

1 /reviews

Zweck der Route:

Gibt eine JSON liste mit allen vorhandenen Skateparks, BilderID's und Obstacles zurück.

Authorization:

Keine

Zusätzliche Parameter

Keine

Body:

Keinen

15. Rest-Routes

15.1.0.25. Response)

Content-Type: JSON

```
1  [
2      {
3          "skateparkId": 1,
4          "name": "Sane Skate Plaza Rum",
5          "longitude": 11.448424,
6          "latitude": 47.26836,
7          "address": "Innsbruck Hallenbad O-Dorf, 6063 Innsbruck",
8          "busstop": "Innsbruck Hallenbad O-Dorf",
9          "rating": 3.5,
10         "obstacles": [
11             {
12                 "ObstacleID": 8,
13                 "Description": "Miniramp",
14                 "difficulty": 4
15             },
16             {
17                 "ObstacleID": 7,
18                 "Description": "Manualpad",
19                 "difficulty": 6
20             },
21             {
22                 "ObstacleID": 2,
23                 "Description": "Flatrail",
24                 "difficulty": 7
25             }
26         ],
27         "pictureIds": [
28             {
29                 "skateparkId": 1,
30                 "picId": 1
31             },
32             {
33                 "skateparkId": 1,
34                 "picId": 2
35             },
36             {
37                 "skateparkId": 1,
38                 "picId": 3
39             },
40             {
41                 "skateparkId": 1,
42                 "picId": 4
43             }
44         ],
45     },
46     ...
47 ]
```

15. Rest-Routes

15.1.0.26. GET /skateparks/:id

```
1   /skateparks/2
```

Zweck der Route

Gibt einen Skatepark mit seinen zugehörigen Obstacles und Bilder ID's aus.

Zusätzliche Parameter id: int (benötigt)

→ Die eindeutige ID des gewünschten Skateparks

Body:

Keinen

Authorization:

Keinen

Response Content-Type: JSON

```
1   [
2     {
3       "skateparkId": 1,
4       "name": "Sane Skate Plaza Rum",
5       "longitude": 11.448424,
6       "latitude": 47.26836,
7       "address": "Innsbruck Hallenbad O-Dorf, 6063 Innsbruck",
8       "busstop": "Innsbruck Hallenbad O-Dorf",
9       "rating": 3.5,
10      "obstacles": [
11        {
12          "ObstacleID": 8,
13          "Description": "Miniramp",
14          "difficulty": 4
15        },
16        {
17          "ObstacleID": 7,
18          "Description": "Manualpad",
19          "difficulty": 6
20        },
21        {
22          "ObstacleID": 2,
23          "Description": "Flatrail",
24          "difficulty": 7
25        }
26      ]
27    ]
28  ]
```

15. Rest-Routes

15.1.0.27. Post /skateparks

```
1   /reviews/2
```

Zweck der Route

Mit dieser Route ist es möglich einen neuen Skatepark hinzuzufügen.

Zusätzliche Parameter `id: int` (benötigt)

→ Die eindeutige ID des gewünschten Skateparks

Body:

Content-Type: JSON

```
1   {
2     "name": "Ibk1",
3     "longitude": 4.23232,
4     "latitude": 2.3232,
5     "address": "dee",
6     "busstop": "dwwd",
7     "obstacles": [
8       {
9         "ObstacleID": 2,
10        "difficulty": 0
11      },
12      {
13        "ObstacleID": 5,
14        "difficulty": 0
15      }
]
```

Authorization:

Keinen

Response Content-Type: JSON

```
1   [
2     {
3       "success": true,
4       "message": "Successfully inserted"
5     }
6   ]
```

15. Rest-Routes

15.1.0.28. DEL /skateparks/

```
1   /users/1
```

Zweck der Route:

Ein User mit einer bestimmten ID wird aus der Datenbank gelöscht.

Authorization:

Bearer

Zusätzliche Parameter id: int (benötigt) → Die eindeutige ID der gewünschten Review.

Body: Keinen

15.1.0.29. Response)

Content-Type: JSON

```
1   {
2     "success": true,
3     "message": "Successfully deleted"
4 }
```

15. Rest-Routes

15.1.0.30. PUT /skateparks/:id

```
1 /skateparks/1
```

Zweck der Route:

Ein Skatepark mit einer bestimmten ID kann nach belieben geändert werden.

Authorization:

Keine

Zusätzliche Parameter id: int (benötigt) → Die eindeutige ID des gewünschten Parks..

Body:

Content-Type: JSON

```
1 {
2     {
3         "Title": "Sane Skate Plaza Rum",
4         "newValue": "Nur Rum"
5     }
6 }
7 }
```

15.1.0.31. Response)

Content-Type: JSON

```
1 {
2     {
3         "success": true,
4         "message": "Successfully updated"
5     }
6 }
```

15. Rest-Routes

15.1.0.32. GET /obstacles

```
1   /obstacles
```

Zweck der Route:

Gibt eine JSON liste mit allen vorhandenen Obstacles zurück.

Authorization:

Keine

Zusätzliche Parameter Keine

Body: Keinen

15.1.0.33. Response)

Content-Type: JSON

```
1   [
2     [
3       {
4         "obstacleId": 1,
5         "description": "Bowl"
6       },
7       {
8         "obstacleId": 2,
9         "description": "box"
10      },
11      {
12        "obstacleId": 3,
13        "description": "Mini Ramp"
14      },
15      ...
16    ]
```

15. Rest-Routes

15.1.0.34. GET /obstacles/:id

```
1   /obstacles/2
```

Zweck der Route

Gibt ein Obstacle mit einer bestimmten ID zurück.

Zusätzliche Parameter `id: int` (benötigt)

→ Die eindeutige ID des gewünschten Obstacles

Body:

Keinen

Authorization:

Keinen

Response Content-Type: JSON

```
1   {  
2       "obstacleId": 2,  
3       "description": "box"  
4   },
```

15. Rest-Routes

15.1.0.35. Post /obstacles

```
1   /obstacles/2
```

Zweck der Route

Mit dieser Route ist es möglich ein neues Obstacle zu erstellen.

Zusätzliche Parameter Keine

Body:

Content-Type: JSON

```
1   "description": "Mini Ramp"
```

Authorization:

Keinen

Response Content-Type: JSON

```
1 [
2   {
3     "success": true,
4     "message": "Successfully inserted"
5   }
6 ]
```

15. Rest-Routes

15.1.0.36. DEL /obstacles/

```
1   /obstacles/1
```

Zweck der Route:

Ein User mit einer bestimmten ID wird aus der Datenbank gelöscht.

Authorization:

Bearer

Zusätzliche Parameter id: int (benötigt) → Die eindeutige ID der gewünschten Review.

Body: Keinen

15.1.0.37. Response)

Content-Type: JSON

```
1   {
2     "success": true,
3     "message": "Successfully deleted"
4 }
```

15. Rest-Routes

15.1.0.38. PUT /skateparks/:id

```
1 /obstacles/1
```

Zweck der Route:

Ein Obstacle mit einer bestimmten ID kann nach belieben geändert werden.

Authorization:

Keine

Zusätzliche Parameter id: int (benötigt) → Die eindeutige ID der gewünschten Review.

Body:

Content-Type: JSON

```
1 {
2     {
3         "Title": "Mini Ramp",
4         "newValue": "Kleine Rampe"
5     }
6 }
7 }
```

15.1.0.39. Response)

Content-Type: JSON

```
1 {
2     {
3         "success": true,
4         "message": "Successfully updated"
5     }
6 }
```

15. Rest-Routes

15.1.0.40. GET /skateparkpictures

```
1   /skateparkpictures
```

Zweck der Route:

Gibt eine JSON liste mit allen vorhandenen Skateparkpicture-ID's zurück.

Authorization:

Keine

Zusätzliche Parameter Keine

Body: Keinen

15.1.0.41. Response)

Content-Type: JSON

```
1   [
2     {
3       "skateparkId": 1,
4       "picId": 1
5     },
6     {
7       "skateparkId": 1,
8       "picId": 2
9     },
10    {
11      "skateparkId": 1,
12      "picId": 3
13    },
14    ...
15  ]
```

15. Rest-Routes

15.1.0.42. GET /skateparkpictures/:id/:pid

1 /obsskateparkpicturestacles/2/1

Zweck der Route

Ruft ein Bild vom Server mit einer Park und Bild Nummer ab. Der Server Antwortet dannach mit den Binärdaten des Bildes.

Zusätzliche Parameter `id: int` (benötigt)

→ Die eindeutige ID des gewünschten Skateparks und des Bildes

Body:

Keinen

Authorization:

Keinen

Response Content-Type: JSON

1 Binaerdaten des Bildes

15. Rest-Routes

15.1.0.43. GET /skateparkpictures/:id

```
1 /s#skateparkpictures/3
```

Zweck der Route

Gibt eine Liste aller Obstacles in einem Park zurück.

Zusätzliche Parameter id: int (benötigt)

→ Die eindeutige ID des gewünschten Skateparks.

Body:

Keinen

Authorization:

Keinen

Response Content-Type: JSON

```
1 [
2     4,
3     6
4 ]
```

15. Rest-Routes

15.1.0.44. GET /check/email

```
1 /check/email
```

Zweck der Route:

Fragt am Server an ob ein User mit der im Body festgelegten Email existiert. Der Server Antwort mit true oder false.

Authorization:

Keine

Zusätzliche Parameter

Keine

Body:

Content-Type: JSON

```
1 {  
2     "email": "test@gmail.com"  
3 }
```

15.1.0.45. Response)

Content-Type: JSON

```
1 { exists: true }
```

Teil VI.

Appendix

Zeitaufwand

Arbeitsprotokoll - Alexander Bertoni

Projekt	Arbeitsaufwand	Prozent
Planung	2:41:10	1.40%
Seite	153:25:45	79.78%
Dokumentation	36:12:34	18.82%
Summe	192:19:29	100.00%

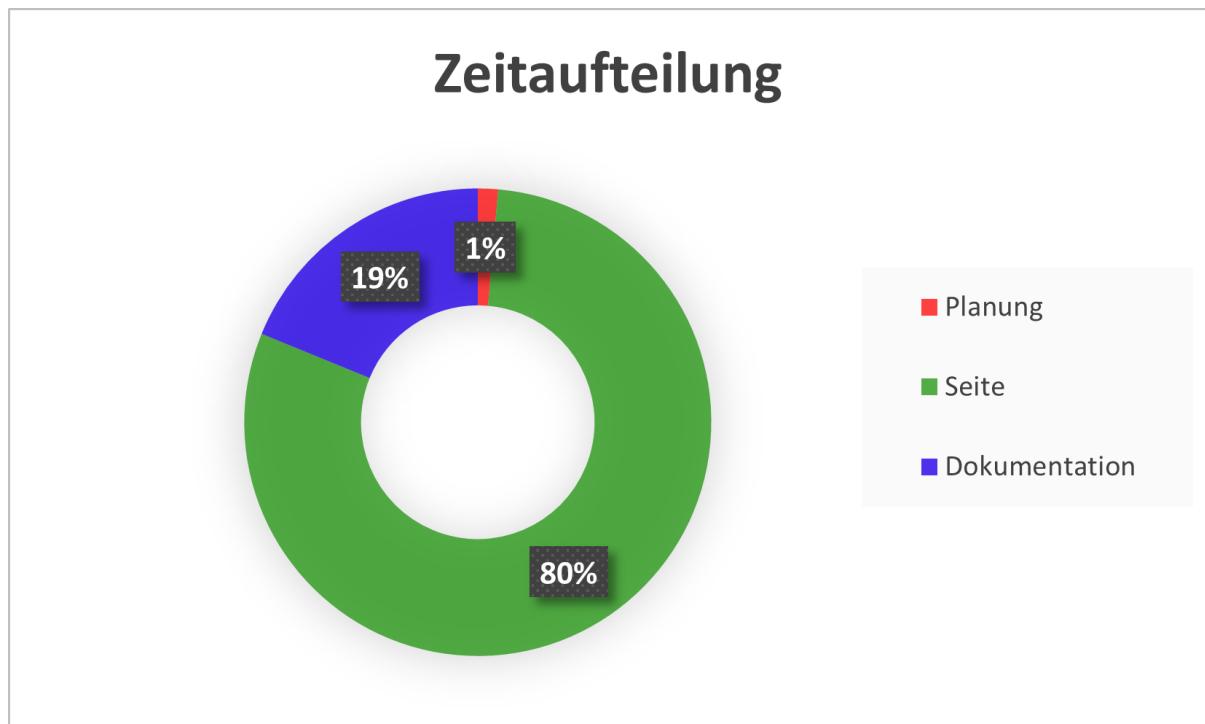


Abbildung 15.1.: Verteilung von Alexanders Arbeitsstunden

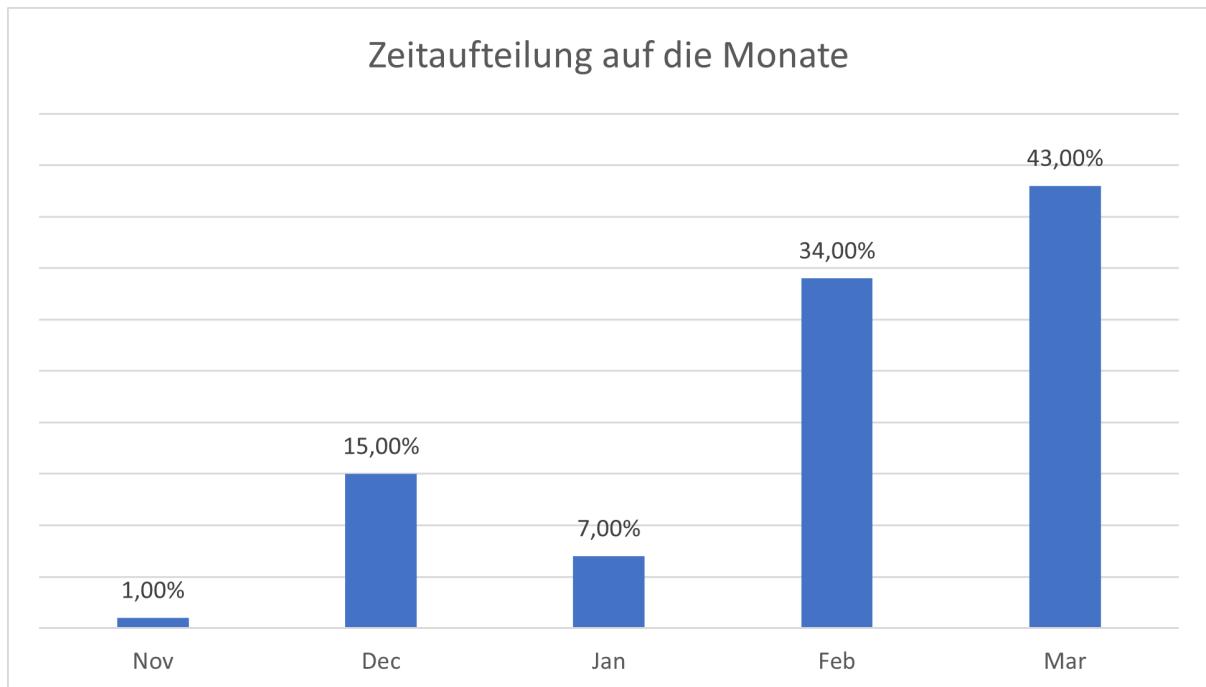


Abbildung 15.2.: Alexanders Arbeitszeiten vom 07.12.2021 bis zum 20.03.2022

Arbeitsprotokoll - Philipp Schuler

Projektteil	Arbeitsaufwand	Prozentueller Anteil
Technologie erlernen	32:57:00	17,16%
Planung	7:31:12	3,92%
App Design und Aufbau	48:35:25	25,30%
Funktionalität	75:25:09	39,27%
Dokumentation	27:34:50	14,36%
Summe	192:03:36	100,00%

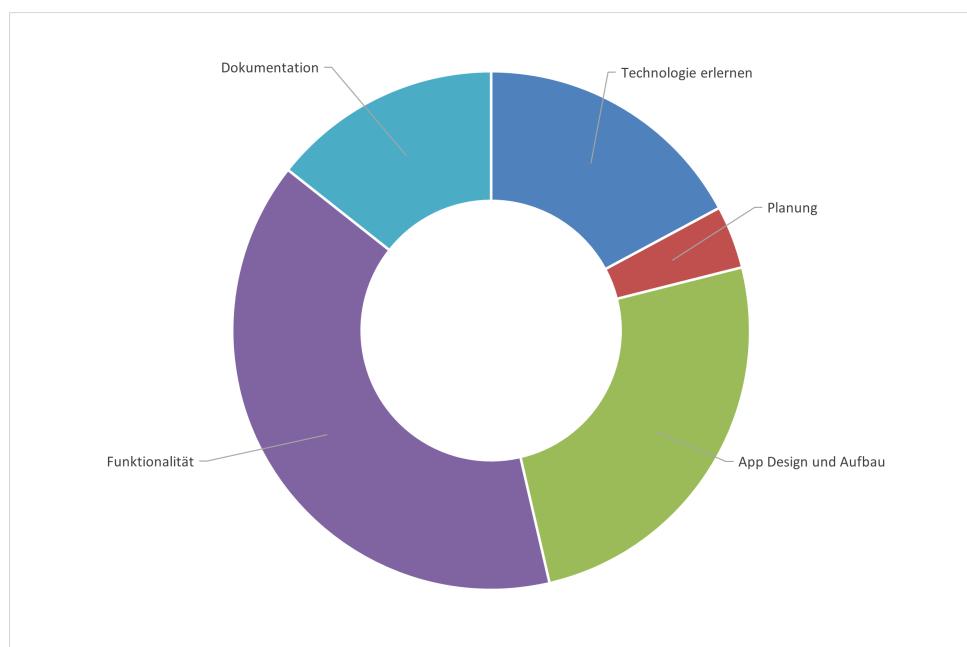


Abbildung 15.3.: Verteilung von Philipp's Arbeitsstunden

Zeitaufteilung auf die Monate

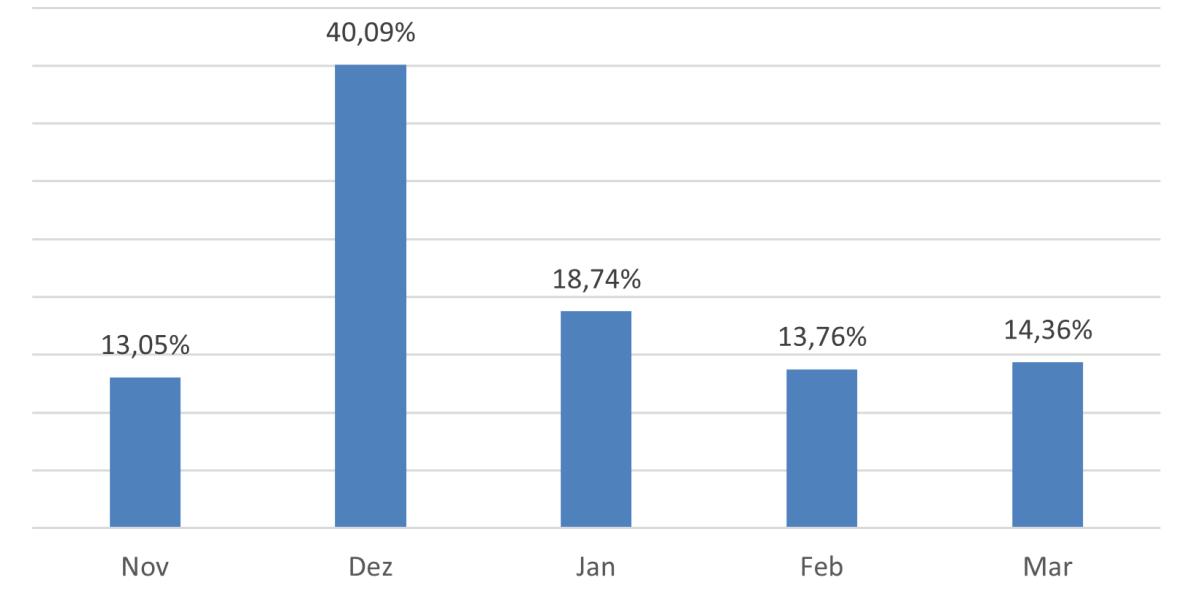


Abbildung 15.4.: Philipp's Arbeitsverlauf vom 30.11.2021 bis zum 20.03.2022

Arbeitsprotokoll - Maximilian Neuner

Projektteil	Arbeitsaufwand	Prozentueller Anteil
Technologie erlernen	28:40:12	15,73%
Planung	8:31:12	4,49%
Backend Entwicklung	103:31:25	57,86%
Dokumentation	37:35:50	20,78%
Summe	178:34:39	100,00%

■ Technologie erlernen ■ Planung ■ Backend Entwicklung ■ Dokumentation

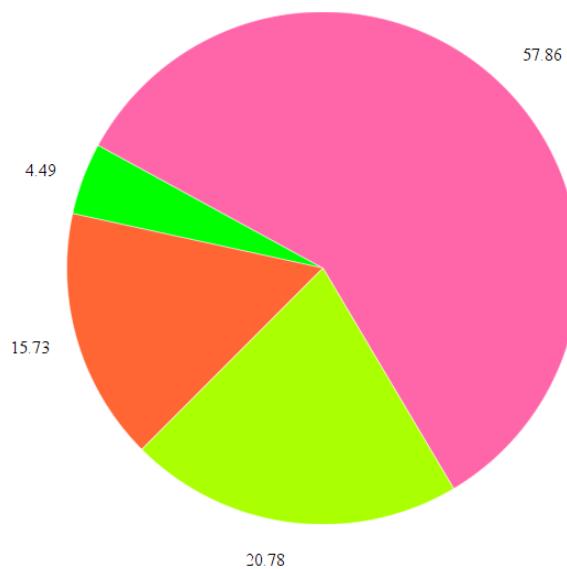


Abbildung 15.5.: Verteilung von Maximilian's Arbeitsstunden

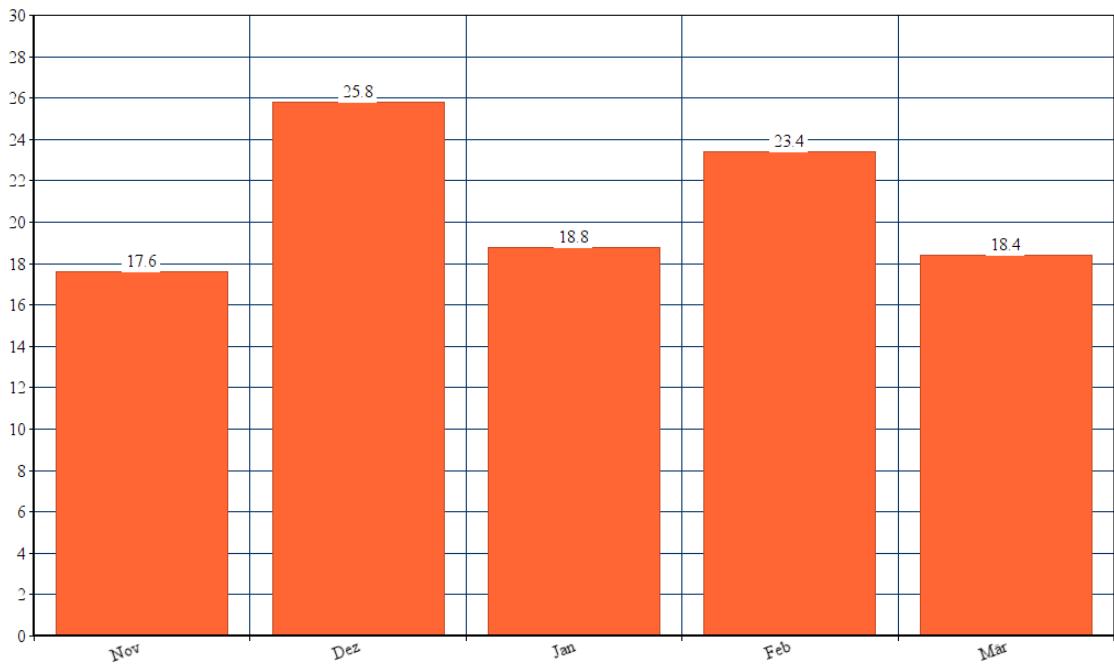


Abbildung 15.6.: Maximilian's Arbeitsverlauf vom 30.11.2021 bis zum 21.03.2022

Literaturverzeichnis

- [1] Adult Swim. Rick and morty staffel 1 werbebild. [https://en.wikipedia.org/wiki/Rick_and_Morty_\(season_1\)](https://en.wikipedia.org/wiki/Rick_and_Morty_(season_1)), 2013. Abgerufen am 08. März 2022.
- [2] Seo-Küche. Webseite. <https://www.seo-kueche.de/lexikon/webseite/>, 2022. Abgerufen am 15. März 2022.
- [3] Polaris Systems. Entstehung und zweck von html. <https://www.polaris-systems.de/tutorials-beitrag/entstehung-und-zweck-von-html.html>, 2009. Abgerufen am 11. März 2022.
- [4] wikibooks. Websiteentwicklung: Css: Geschichte. https://de.wikibooks.org/wiki/Websiteentwicklung:_CSS:_Geschichte, 2013. Abgerufen am 13. März 2022.
- [5] mdn web docs. Javascript. <https://developer.mozilla.org/de/docs/Web/JavaScript>, 2022. Abgerufen am 13. März 2022.
- [6] mdn web docs. Funktionen erster klasse. https://developer.mozilla.org/de/docs/Glossary/First-class_Function, 2022. Abgerufen am 13. März 2022.
- [7] computerweekly. Objektorientierte programmierung (oop). <https://www.computerweekly.com/de/definition/Objektorientierte-Programmierung-OOP>, 2022. Abgerufen am 13. März 2022.
- [8] Facebook Open Source. React. <https://de.reactjs.org/>, 2013. Abgerufen am 10. März 2022.
- [9] React. Virtual dom and internals. <https://reactjs.org/docs/faq-internals.html>, 2022. Abgerufen am 14. März 2022.
- [10] digital pioneers. Was ist eigentlich react? <https://t3n.de/news/react-facebook-623999/>, 2022. Abgerufen am 14. März 2022.
- [11] Brickmakers. React — eine einföhrung in fünf minuten. https://blog.brickmakers.de/react-eine-einf%C3%BChrung-in-f%C3%BCnf-minuten?utm_campaign=Digitale%20Transformation&utm_source=Medium&utm_medium=Medium%20React%20Einf%C3%BChrung, 2017. Abgerufen am 14. März 2022.
- [12] envatotuts+. Stateful vs. stateless funktionsteilen in react. <https://code.tutsplus.com/de/tutorials/stateful-vs-stateless-functional-components-in-react--cms-29541>, 2022. Abgerufen am 14. März 2022.
- [13] React. Introducing jsx. <https://reactjs.org/docs/introducing-jsx.html>, 2022. Abgerufen am 14. März 2022.

Literaturverzeichnis

- [14] React. Komponenten und props. <https://de.reactjs.org/docs/components-and-props.html>, 2022. Abgerufen am 14. März 2022.
- [15] Frederik Venzke. Angular vs. react - ein ausführlicher vergleich. welche bibliothek passt zu ihrem projekt? <https://www.ventzke-media.de/blog/angular-vs-react-vergleich.html#Vergleich>, 2020. Abgerufen am 15. März 2022.
- [16] Statcounter. Mobile operating system market share. <https://gs.statcounter.com/os-market-share/mobile/worldwide>, 2022. Abgerufen am 14. März 2022.
- [17] Google. Android open source project. <https://source.android.com>, 2022. Abgerufen am 14. März 2022.
- [18] Encyclopedia Britannica. Android history. <https://www.britannica.com/technology/Android-operating-system>, 2020. Abgerufen am 14. März 2022.
- [19] Wikipedia. React native history. https://en.wikipedia.org/wiki/React_Native#History, 2022. Abgerufen am 15. März 2022.
- [20] Christina Warren. Facebook new ios app. <https://mashable.com/archive/facebook-ios-5>, 2012. Abgerufen am 15. März 2022.
- [21] Facebook Open Source. React native showcase. <https://reactnative.dev/showcase>, 2022. Abgerufen am 15. März 2022.
- [22] Fanghao Chen. Why discord is sticking with react native. <https://blog.discord.com/why-discord-is-sticking-with-react-native-ccc34be0d427>, 2018. Abgerufen am 18. März 2022.
- [23] Google LLC. Flutter. <https://flutter.dev>, 2022. Abgerufen am 20. März 2022.
- [24] Microsoft Corporation. Xamarin. <https://visualstudio.microsoft.com/de/xamarin/>, 2022. Abgerufen am 20. März 2022.
- [25] Ionos. Was ist ein server. <https://www.ionos.at/digitalguide/server/knowhow/was-ist-ein-server-ein-begriff-zwei-definitionen/>, 2020.
- [26] arocom. Nodejs erklärung. <https://www.arocom.de/fachbegriffe/webentwicklung/nodejs>, 2022. Abgerufen am 16. März 2022.
- [27] DEV-Insider. Nodejs erklärung2. <https://www.dev-insider.de/was-ist-nodejs-a-972703/>, 2002.
- [28] Oracle. Was ist eine datenbank. <https://www.oracle.com/de/database/what-is-database/>, 2021.
- [29] datenschutzexperte. Wie funktioniert hashing. <https://www.datenschutzexperte.de/blog/datenschutz-im-unternehmen/passwort-hashing-salted-password-hashing/>, 2022.
- [30] Kinsta. Was ist mysql bruh. <https://kinsta.com/de/wissensdatenbank/was-ist-mysql/>, 2020.

Literaturverzeichnis

- [31] Red Hat. Docker funktionsweiße. <https://www.redhat.com/de/topics/containers/what-is-docker#:~:text=Die%20Docker%2DTechologie%20verwendet%20den,~getrennt%20voneinander%20betreiben%20zu%20k%C3%B6nnen.,> 2022.
- [32] Infinite Red Inc. Ignite cli for react native. <https://github.com/infinited/ignite>, 2022. Abgerufen am 18. März 2022.
- [33] Expo. Create-react-native-app. <https://github.com/expo/create-react-native-app>, 2022. Abgerufen am 18. März 2022.
- [34] Facebook Open Source. React native core components. <https://reactnative.dev/docs/intro-react-native-components>, 2022. Abgerufen am 17. März 2022.
- [35] Datenbanken-verstehen.de. Camel case notation. <https://www.datenbanken-verstehen.de/namenskonventionen-datenbank/camel-case-notation/>, 2022. Abgerufen am 17. März 2022.
- [36] Samantha Ming. Javascript module cheatsheet. <https://medium.com/dailyjs/javascript-module-cheatsheet-7bd474f1d829>, 2019. Abgerufen am 17. März 2022.
- [37] Expo. Expo cli for react native. <https://docs.expo.dev>, 2022. Abgerufen am 18. März 2022.
- [38] itfoundry. Poppins font. <https://github.com/itfoundry/Poppins>, 2019. Abgerufen am 19. März 2022.
- [39] Color-name.com. Color-name.com. <https://www.color-name.com/jade.color>, 2022. Abgerufen am 19. März 2022.
- [40] Michaela Lavicka. Mycolor.space. <https://mycolor.space/?hex=%2300A86B&sub=1>, 2022. Abgerufen am 19. März 2022.
- [41] Joel Arvidsson. React native vector icons. <https://github.com/oblador/react-native-vector-icons>, 2022. Abgerufen am 20. März 2022.
- [42] Dan Abramov. Redux toolkit. <https://redux-toolkit.js.org/introduction/getting-started#purpose>, 2022. Abgerufen am 20. März 2022.
- [43] Ionos. Json web token. <https://www.ionos.at/digitalguide/websites/web-entwicklung/json-web-token-jwt-vorgestellt/>, 2022.
- [44] Facebook Open Source. React native github. https://en.wikipedia.org/wiki/React_Native#History, 2022. Abgerufen am 15. März 2022.
- [45] Facebook Open Source. React native cli setup. <https://reactnative.dev/docs/environment-setup>, 2022. Abgerufen am 18. März 2022.
- [46] Dan Abramov. Redux toolkit. <https://redux-toolkit.js.org/introduction/getting-started#purpose>, 2022. Abgerufen am 20. März 2022.

Abbildungsverzeichnis

1.1.	Vergleich zwischen Portal aus der Serie und dem Logo	13
3.1.	ReactJS Logo [8]	19
3.2.	Ergebnis Renderfunktion	21
6.1.	Fehlermeldung wenn das Daten holen vom Server fehlschlägt	45
6.2.	Navbar der Webseite wenn man nicht eingeloggt ist	48
6.3.	Navbar der Webseite wenn man eingeloggt ist	48
6.4.	Die Fußzeile der Webseite	49
6.5.	Die Startseite	52
6.6.	CSS Boxen	53
7.1.	Die Startseite	54
7.2.	Fehlermeldung für nicht Übereinstimmende Passwörter	55
7.3.	Fehlermeldung für nicht Übereinstimmende Passwörter	56
7.4.	Login Page mit verstecktem Passwort	57
7.5.	Login Page mit gezeigtem Passwort	58
7.6.	Augensymbol zum Umschalten	59
7.7.	Profil	59
7.8.	Liste der Parks	60
7.9.	Liste der Parks	61
7.10.	Einen Vorschlag erstellen	61
7.11.	Liste der Vorschläge	62
7.12.	Park erstellen	63
7.13.	Die Startseite	64
7.14.	Die Startseite	65
7.15.	Eine geschriebene Bewertung	65
9.1.	Tabelle - Core Components und deren Gegenstücke [34]	72
9.2.	Der Metro-Server in Aktion	74
9.3.	Die Beispiel-App in Android Studio	75
9.4.	Expo beim Start	78
9.5.	Webinterface von Expo	79
9.6.	Die weiße Leinwand von Expo	80
10.1.	Beispiel für ein Snippet	82
10.2.	Vom Snippet erzeugter Code	82
10.3.	Der TextInput für die Email am Login-Screen	85
10.4.	TextInput ausgewählt	85
10.5.	Fehlerhafte Eingabe	85
10.6.	Korrekte Eingabe	85

Abbildungsverzeichnis

10.7. Login-Informationen falsch	87
10.8. Login-Informationen richtig	87
11.1. Distanz zwischen Hall in Tirol und Skatepark USI Innsbruck	89
11.2. Die Berechtigungs-Anfrage beim Starten der App	91
11.3. Beim Ablehnen der Aufforderung wird die Anfrage an die Directions-API abgebrochen	92
13.1. Stack Navigator und Tab-Navigator	102
13.2. Alle Skateparks werden an der korrekten Stelle angezeigt	104
13.3. Ein ausgewählter Marker	107
13.4. Die Profil-Ansicht	108
13.5. SkateparksList Screen	110
13.6. Suchfunktion Demonstration	111
13.7. Sortieren nach Zeit	112
13.8. Adresse und nächste Haltestelle werden als erstes gezeigt	113
13.9. Zu jedem Park werden auch Hindernisse mit Schwierigkeitseinschätzungen abgespeichert	114
13.10 Daten eintragen und absenden	115
13.11 Alle Bewertungen	116
13.12 Daten eintragen und absenden	117
13.13 Daten eintragen und absenden	118
13.14 Daten eintragen und absenden	119
15.1. Verteilung von Alexanders Arbeitsstunden	155
15.2. Alexanders Arbeitszeiten vom 07.12.2021 bis zum 20.03.2022	156
15.3. Verteilung von Philipp's Arbeitsstunden	157
15.4. Philipp's Arbeitsverlauf vom 30.11.2021 bis zum 20.03.2022	158
15.5. Verteilung von Maximilian's Arbeitsstunden	159
15.6. Maximilian's Arbeitsverlauf vom 30.11.2021 bis zum 21.03.2022	160

List of Codes

5.1. Code-Snippet-Hashing	39
9.1. CMD - React Native CLI init	69
9.2. React Component - App.js	71
9.3. CMD - Kurzschreibweise für den Befehl npm run start	74
9.4. CMD - Die App wird auf dem Emulator installiert und ausgeführt.	75
9.5. CMD - Erstellung eines TCP-Tunnels über USB	76
9.6. CMD - Als Beispiel-Projekt ist das Template blank geeignet.	77
9.7. CMD - Man wird bei einer Eingabeaufforderung darüber informiert, dass dieser Prozess nicht rückgängig gemacht werden kann.	80
10.1. React Component - Snippet rnfec in der Datei Test.js	82
10.2. React Component - Custom Text Komponente	83
10.3. React Component - Auch der Button-Component von React Native lässt sich leicht nachbauen.	84
10.4. React Component - FlashMessage-Komponente in root/App.js	87
11.1. JavaScript Funktion - useFetch mit Aktualisierungsfunktion	88
11.2. JavaScript Funktion - Die Url für den API-Request wird generiert.	89
11.3. JavaScript Funktion - Berechtigung prüfen.	90
11.4. JavaScript Funktion - Berechtigung anfragen.	90
11.5. JavaScript Funktion - Geolocation-API liefert die Position.	93
12.1. React Component - AuthProvider ist die äußerste Komponente.	94
12.2. JavaScript Funktion - Aus dem Hook werden zwei Variablen extrahiert, state und dispatch.	95
12.3. JavaScript Funktion - Die State-Veränderung "SIGN-IN" wird aufgerufen.	96
12.4. JavaScript Funktion - Die Auth-Funktionen	97
12.5. React Component - Alle Kinder erhalten Zugriff auf authContext und state	98
12.6. React Component - Ob ein Benutzer eingeloggt ist, hängt von state.userToken ab.	100
13.1. JavaScript Funktion - In NavigationContainer können Navigationen erstellt werden.	102
13.2. React Component - Bottom-Tab-Navigation	103
13.3. React Component - Der Karten-Tab	105
13.4. React Component - Die MapView-Komponente aus react-native-maps	106
14.1. Code-Snippet-Authorization	124