



ulm university universität
ulm

Universität Ulm | 89069 Ulm | Germany

Fakultät für
Ingenieurwissenschaften,
Informatik und
Psychologie
Institut für Datenbanken
und Informationssysteme

Mobile Application Development

Ausarbeitung zur App "Coaster2Go" an der Universität Ulm

Vorgelegt von:

Fabian Fischbach, Luis Beaucamp und Tim Stenzel

Gutachter:

Marc Schickler

Betreuer:

Marc Schickler

2017

Fassung 30. August 2017

© 2017 Fabian Fischbach, Luis Beucamp und Tim Stenzel

This work is licensed under the Creative Commons. Attribution-NonCommercial-ShareAlike 3.0 License. To view a copy of this license, visit <http://creativecommons.org/licenses/by-nc-sa/3.0/de/> or send a letter to Creative Commons, 543 Howard Street, 5th Floor, San Francisco, California, 94105, USA.

Satz: PDF-L^AT_EX 2_ε

Inhaltsverzeichnis

1 Einleitung	1
1.1 Motivation und Problemstellung	1
1.2 Zielsetzung	2
1.3 Struktur der Arbeit	3
2 Grundlagen	5
2.1 Grundlagen der Bewertungen und Wartezeitenberechnung	5
2.2 Mobile Plattform Android	7
2.3 Frameworks	9
3 Anforderungsanalyse	11
3.1 Funktionale Anforderungen	11
3.2 Nichtfunktionale Anforderungen	13
4 Konzept, Entwurf und Implementierung	15
4.1 Implementierungsdetails	15
4.2 Besonderheiten	22
4.2.1 Wartezeit berechung und -darstellung	22
4.2.2 Client-Server-Kommunikation mit Azure	24
4.2.3 Nutzerverwaltung mit Firebase	27
4.3 Architektur	30
4.3.1 Datenmodell und -speicherung	30
4.3.2 Klassen- und Activity-Architektur	31
4.3.3 Arbeitsaufteilung	31
4.4 Herausforderungen während der Implementierung	33
5 Anforderungsabgleich	35
5.1 Funktionale Anforderungen	36
5.2 Nichtfunktionale Anforderungen	36
6 Ausblick	37

Inhaltsverzeichnis

7 Fazit

39

1

Einleitung

In dieser Dokumentation wird die Entwicklung einer App im Rahmen des Anwendungsfaches Mobile Application Development an der Universität Ulm und die dadurch erstellte App in Form einer Freizeitpark-App vorgestellt.

1.1 Motivation und Problemstellung

Die Problemstellung war für dieses Projekt wesentlich offener gegeben, da wir eine beliebige App für unsere präferierte mobile Platform entwickeln sollten. Nachdem wir einige Ideen gesammelt hatten, uns Gedanken zur Implementierung gemacht haben und die Vor- und Nachteile abgewägt hatten, entschieden wir uns für eine Freizeitpark-App. Als zentrale Platform für Freizeitparkliebhaber sollten Parks mit Standorten, zugehörige Attraktionen mit Wartezeiten und Bewertungen verfügbar sein.

Eine große Anzahl potentieller Nutzer von Freizeitpark-Apps dürfte vorhanden sein. So hat zum Beispiel alleine der Europapark, Deutschlands größter Freizeitpark, jährlich 5,5 Millionen Besucher, Tendenz steigend¹. Zusammen mit allen anderen Freizeitparks in Deutschland oder sogar weltweit kommt schnell eine beachtliche Summe von Freizeitparkbesuchern und damit potentiellen Nutzern zusammen, die wir mit unserer App ansprechen wollen. Umso erstaunlicher ist es, dass bisher nicht wirklich Apps oder Dienste für Freizeitparkbesucher zur Verfügung stehen.

Wenn man einmal die aktuelle Marktlage betrachtet fällt auf, dass es nur Apps für einzelne bestimmte große Parks, wie zum Beispiel Disneyworld, eigens entwickelte Apps

¹Quelle: <http://presse.europapark.com/de/presse/nachricht/datum/2017/07/21/europa-park-ist-tourismusmagnet-im-schwarzwald/>

1 Einleitung

oder von Fans entwickelte Apps gibt. Bei genauerer Betrachtung zeigen diese Apps aber Mängel auf. Sie bieten kaum oder nur wenig Funktionen, sind teilweise recht veraltet und nicht benutzerfreundlich designt. Wir konnten keine App für Android finden, welche im Ansatz der unserer Vorstellung ähnelte.



Abb. 1.1: Symbolbild: Potentielle Nutzer²

1.2 Zielsetzung

Da also auf dem Markt eine Nachfrage nach einer Freizeitpark-App entsteht, wollen wir diese ausnutzen um im Laufe dieses Projekts eine eigene Anwendung zu entwickeln. Diese wollen wir auf Basis von Android und Java implementieren. Dabei geht es uns primär darum, den Umgang mit den neuen Techniken (beispielsweise GPS) zu erlernen und unsere bereits vorhandenen Kenntnisse zu erweitern.

Inhaltlich möchten wir eine Anwendung entwickeln in der es möglich sein sollte aus einer Liste von Freizeitparks, deren Attraktionen anzusehen. Inklusive zu den Attraktionen bieten wir die zugehörigen Wartezeiten, Statistiken und Bewertungen. Ebenfalls sollte es die App dem Benutzer ermöglichen selbst Wartezeiten einzutragen und Parks bzw.

²Bildquelle: [https://de.wikipedia.org/wiki/Datei:Europa-Park_-_Blue_Fire_Megacoaster_\(32\).JPG](https://de.wikipedia.org/wiki/Datei:Europa-Park_-_Blue_Fire_Megacoaster_(32).JPG)

die Attraktionen zu bewerten. Außerdem sollen in einem Admin-Bereich die Funktionen zur Erstellung und Bearbeitung eigener Parks und Attraktionen vorhanden sein. Des Weiteren stehen auch die Benutzerfreundlichkeit, das Design und die Kompatibilität mit möglichst vielen Android-Smartphones im Vordergrund, sodass ein breites Spektrum an Nutzern angesprochen wird.

1.3 Struktur der Arbeit

In dieser Dokumentation werden zuerst einmal die Grundlagen der Bewertungen und Wartezeitenberechnung erklärt, sowie Android vorgestellt und unsere verwendeten Frameworks präsentiert. Den Hauptteil bildet die Implementierung, welche unsere App im Allgemeinen und mit ihren Besonderheiten vorstellt. Außerdem wird darin unsere Architektur gezeigt und wir erläutern Schwierigkeiten, die wir während der Implementierungsphase hatten. Abschließend gibt es einen Anforderungsabgleich und einen Ausblick auf die Zukunft des Projektes.

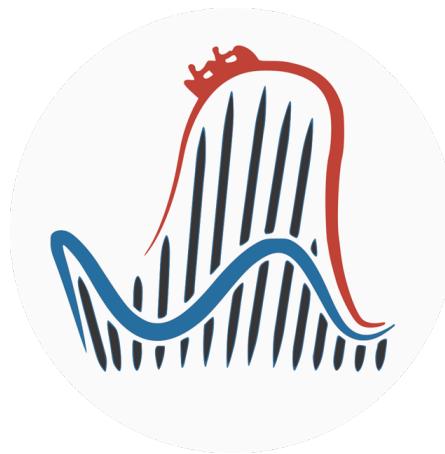


Abb. 1.2: Coaster2go Logo

2

Grundlagen

2.1 Grundlagen der Bewertungen und Wartezeitenberechnung

Da Freizeitparks mit sehr großen Besucherströmen zu tun haben, entstehen schnell lange Schlangen von Menschen an den verschiedensten Stellen, wie Attraktionen oder Essensausgaben, im Park. Die Zeit, die ein Besucher an einer Attraktion mit Warten verbringt, bezeichnet man als Wartezeit der Attraktion. Diese kann vom einstelligen bis hin zu einem dreistelligen Minutenbereich andauern. Größere Freizeitparks sind mittlerweile sehr gut darin, Besucherzahlen abzuschätzen und daraus aktuelle oder durchschnittliche Wartezeiten anzugeben. Diese werden meist am Eingang einer Attraktion angegeben.



Abb. 2.1: Wartezeitanzeige mit geringer (links) und hoher Wartezeit (rechts)¹

Der Nachteil daran ist, dass die Wartezeiten nur direkt beim Betreten der Attraktion sichtbar sind. Es wäre für den Besucher besser, diese auch andernorts und vergleichend betrachten zu können, um besser voraus planen zu können. Auch dafür haben einige

¹Bildquellen: <http://photobucket.com/gallery/user/Parksonline/media/cGF0aDovMTUtMV96cHM2NjdiYWYxYy5qcGc=/?ref=>, https://www.coasterfriends.de/forum/attachments/eure-tripreports-untere-freude/240493d1382869532-europa-park-am-26-10-2013-dsc_4001.jpg

2 Grundlagen

Parks eine Lösung gefunden. So gibt es zum Beispiel Monitore mit allen Wartezeiten im Park oder eine parkeigene App mit allen aktuellen Wartezeiten.

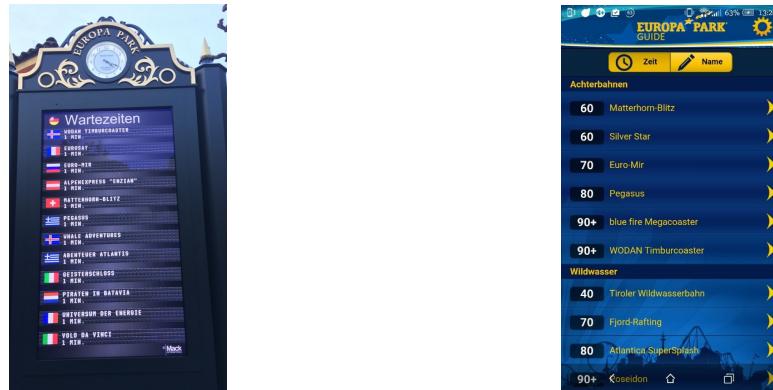


Abb. 2.2: Wartezeitübersicht des Europaparks als Monitor (links) und als App (rechts)²

Da diese parkeigenen Apps, aus verschiedenen Gründen, nur innerhalb des Parks funktionieren, bleibt auch hier das Problem, dass die aktuellen Wartezeiten nur innerhalb des Parks verfügbar sind und man nicht von außerhalb seinen Freizeitparkbesuch planen kann. Es kommt hinzu, dass man meistens nur eine aktuelle Wartezeit zur Verfügung hat, gerne aber weitere Informationen wie die Durchschnitte bestimmter Tage oder Uhrzeiten hätte. Außerdem gibt es bisher kein direktes Feedback der Nutzer über ihre tatsächlich verbrachte Wartezeit.

Um all diese Probleme zu behben gibt es in unserer App verschiedene Metriken zum Vergleichen der Wartezeiten verschiedener Attraktionen. Diese werden nicht von den Freizeitparks zur Verfügung gestellt, sondern von den Nutzern selbst eingetragen, um somit aus der Gesamtheit aller Nutzerdaten verschiedene Werte berechnen zu können. Speziell unterscheidet unsere App zwischen folgenden fünf Arten von Wartezeiten einer Attraktion:

- Aktuelle Wartezeit: der Durchschnitt der neusten eingetragenen Wartezeiten einer Attraktion, denkbar wäre hier auch das Einbinden von Livedaten aus vorhandenen APIs bestimmter Freizeitparks

²Bildquellen: <https://www.ep-board.de/download/file.php?id=31185&mode=view>, <http://uploads.tapatalk-cdn.com/20160506/d8e6e2c4c8bfc31c9dee0dc32399515a.jpg>

2.2 Mobile Platform Android

- Tagesdurchschnitt: der Durchschnitt aller eingetragenen Wartezeiten des aktuellen Tages einer Attraktion
- Gesamtdurchschnitt: der Durchschnitt aller eingetragenen Wartezeiten einer Attraktion
- Durchschnitt nach Uhrzeit: der Durchschnitt aller eingetragenen Wartezeiten einer Attraktion eines bestimmten Zeitraums
- Wartezeitenliste: das Anzeigen aller eingetragenen Wartezeiten, welche nach Datum oder Dauer sortiert werden können

Da nicht nur die Wartezeit sondern auch die Qualität einer Attraktion ausschlaggebend ist, gibt es in unserer App zusätzlich ein Bewertungssystem für die einzelnen Attraktionen und auch für den Park insgesamt. Dieses ist wie jedes klassische Bewertungssystem durch das "5-Sterne-Bewertungssystem" realisiert. Dabei entsprechen 5 Sterne einer hervorragenden und 0 Sterne einer mangelhaften Bewertung.

2.2 Mobile Platform Android

Android ist ein mobiles Betriebssystem, also für Smartphones und Tablets, das von Google entwickelt wurde und auf Linux basiert. Die App-Entwicklung ist geprägt durch einzelne Aktivitäten (ein angezeigter Screen ist eine Aktivität), die miteinander kommunizieren und in ihrer 'Lebenszeit' ein vorgegebenes Zustandmodell 2.3 durchlaufen. Wir beschränken uns in Bezug auf die Plattform Android auf diese kurze Einführung, da die Plattform dank seiner weltweit hohen Verbreitung als bekannt angesehen werden dürfte.

Dieses Zustandmodell ist auch anfangs einer der Nachteile von Android, da es nicht so leicht zu verstehen ist und uns auch ein paar Probleme bereitet hat. Nachdem wir uns aber im Laufe der App-Entwicklung immer mehr mit Android vertraut gemacht haben, war auch das Modell kein Problem mehr, sondern eher ein Vorteil, da es sehr logisch und durchdacht ist. Eine weitere Schwierigkeit, die während der Entwicklung aufgetreten ist, ist, dass es so viele verschiedene Android-Versionen und Geräte gibt. Da wir unsere App für so viele Versionen wie möglich entwickeln wollten, kamen deshalb auch mal

2 Grundlagen

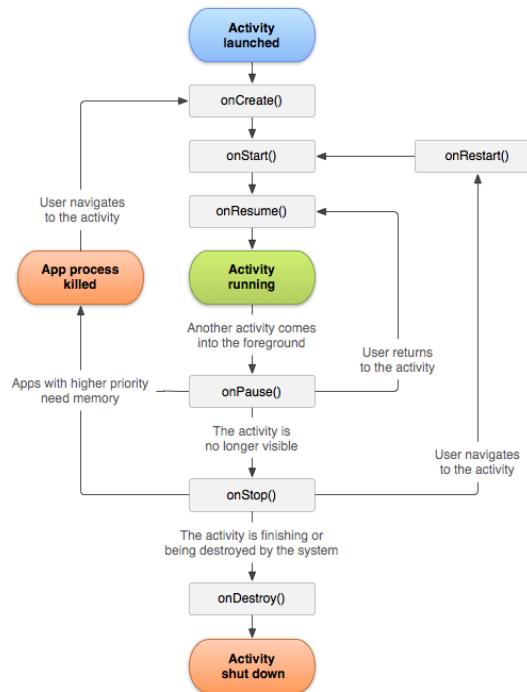


Abb. 2.3: Android Zustandsmodell³

das ein oder andere Problem auf, wie z. B. dass manche Libraries oder Frameworks erst ab bestimmten Versionen verfügbar sind oder die vielen verschiedenen Geräte alle unterschiedlichen Seiten- und Größenverhältnisse haben.

Die Vorteile von Android überwiegen aber auf jeden Fall, vor allem, wenn man sich damit tiefer beschäftigt und eingearbeitet hat. Einer der größten Vorteile ist die sehr gute Dokumentation von Android durch die das Einlesen in die Möglichkeiten und Funktionen recht leicht ist. Auch die weltweite Verbreitung und Beliebtheit von Android ist hier ein Vorteil, da es sehr viele Entwickler gibt und so jedes Problem schon einmal aufgetreten ist und daher auch zu den allermeisten auch Lösungen oder Workarounds bekannt sind. Außerdem wird Android stetig Weiterentwickelt, weshalb immer mehr möglich ist und der Umgang mit bestimmten Funktionen, wie z. B. der Zugriff auf Gerätefunktionen wie Kamera oder GPS immer leichter wird. Weitere Vorteile für uns sind die Vertrautheit mit

³Bildquelle: https://developer.android.com/images/activity_lifecycle.png

Java und die Einfachheit von Android Studio, welches wir für die Entwicklung unserer App benutzt haben.

2.3 Frameworks

Frameworks sind eine Art Gerüst oder Rahmen, die eingesetzt werden um das Programmieren zu vereinfachen und die geschriebenen Zeilen zu verringern. Wir haben in unserer App die folgenden aufgelisteten Frameworks als Hilfen genutzt. Alle sind öffentlich und über den jeweiligen Namen auffindbar.

- Microsoft Azure Mobile App Service: Server mit persistenter Speicherung in SQLite-Datenbank (näheres dazu in Kapitel 4.2.2)
- Google Firebase: einfache Nutzerverwaltung (näheres dazu in Kapitel 4.2.3)
- Cloudinary: kostenloses Hosten und Auslesen von Bildern via HTTP-Request
- Picasso: erlaubt einfaches Handling (z. B. Größentransformationen) von Bildern in oftmals einer Zeile
- Glide: ermöglicht eine bessere Darstellung der animierten Lade-Animation
- MPAndroidCharts: ermöglicht die Erstellung von Diagrammen (in unserem Fall Säulendiagramme für die Statistiken)
- MaterialChipsInput: Darstellung der Unterkategorien als Chips

3

Anforderungsanalyse

In diesem Kapitel werden die ursprünglich definierten funktionalen und nichtfunktionalen Anforderungen tabellarisch dargestellt. Jede der Anforderungen hat einen eindeutigen Identifikator (ID), einen Titel (TITEL), und eine Beschreibung (BES).

3.1 Funktionale Anforderungen

ID:	FA1
TITEL:	Startbildschirm
BES:	Nach dem Start der Anwendung sieht der Benutzer einen Startbildschirm, auf dem er sich anmelden bzw. registrieren kann, oder ohne Anmeldung zur Parkübersicht gelangt.
ID:	FA2
TITEL:	Parkübersicht
BES:	Diese Ansicht zeigt eine Liste aller Parks, die auf verschiedene Weisen sortiert werden kann. Außerdem gibt es eine getrennte Liste, in der sich die favorisierten Parks des Nutzers befinden. Sofern die Position des Smartphones bekannt ist, wird die Entfernung neben den Parks angezeigt und die Liste kann nach Entfernung sortiert werden.
ID:	FA3
TITEL:	Park Detailansicht
BES:	Für jeden Park gibt es eine Detailansicht mit näheren Informationen. Hier befinden sich Links/Buttons zu den Bewertungen, der Kartenansicht, sowie zur Attraktionsübersicht des Parks. Außerdem kann der Park zu den Favoriten hinzugefügt werden.

3 Anforderungsanalyse

ID:	FA4
TITEL:	Bewertungsseite (Park / Attraktionen)
BES:	Es gibt eine Ansicht der neuesten Bewertungen (nach Datum sortiert). Eine Bewertung hat jeweils Verfasser, Datum, Stern-Anzahl, und Kommentar. Falls der Benutzer eingeloggt ist, sieht er einen Button zum Hinzufügen einer Bewertung.
ID:	FA5
TITEL:	Bewertung verfassen
BES:	Sofern eingeloggt, kann der Benutzer in einem separaten Dialog eine Bewertung verfassen. Hat er zuvor schon eine Bewertung zu diesem Park / dieser Attraktion verfasst, kann er diese bearbeiten. Der Administrator kann den Kommentar ausblenden.
ID:	FA6
TITEL:	Attraktionsübersicht
BES:	Diese Ansicht zeigt eine Liste aller Attraktionen, die auf verschiedene Weisen sortiert und gefiltert werden kann. Außerdem gibt es eine getrennte Liste, in der sich die favorisierten Attraktionen des Nutzers befinden. Neben jeder Attraktion wird, sofern verfügbar, die jeweilige „aktuelle“ Wartezeit angezeigt. (Sofern die Position des Smartphones bekannt ist, wird die Entfernung neben den Parks angezeigt und die Liste kann nach Entfernung sortiert werden.)
ID:	FA7
TITEL:	Attraktion Detailansicht
BES:	Für jede Attraktion gibt es eine Detailansicht mit näheren Informationen. Hier befinden sich Links/Buttons zu den Bewertungen, der Kartenansicht, sowie zu den Wartezeiten. Auch kann die Attraktion zu den Favoriten hinzugefügt werden. Zudem gibt es verschiedene Statistiken zu den Wartezeiten (siehe FA8). Ist der Nutzer eingeloggt und befindet sich in der Nähe des Parks (GPS muss aktiviert sein), so kann er eine Wartezeit eintragen. Dies kann er nur ein Mal pro Stunde machen.

3.2 Nichtfunktionale Anforderungen

ID:	FA8
TITEL:	Wartezeit-Statistiken
BES:	<p>Es gibt vier verschiedene Statistiken über die Wartezeit bei einer Attraktion:</p> <ul style="list-style-type: none"> • „Aktuell“: Zeigt die aktuelle Wartezeit an • „Heute“: Zeigt den Durchschnitt für den aktuellen Tag an • „Gesamt“: Zeigt den bisherigen Gesamtdurchschnitt an • Ein Säulendiagramm, welches den stündlichen bisherigen Durchschnitt anzeigt
ID:	FA9
TITEL:	Wartezeiten-Verlauf
BES:	Es gibt eine Listenansicht der bisher eingetragenen Wartezeiten einer Attraktion. Ein Wartezeiten-Eintrag enthält Verfasser, Datum, Uhrzeit, und Wartezeit (in Minuten).
ID:	FA10
TITEL:	Admin-Funktionen
BES:	<p>Ein Administrator hat folgende Zusatzfunktionen:</p> <ul style="list-style-type: none"> • Parks / Attraktionen hinzufügen und erstellte bearbeiten • Kommentare ausblenden

3.2 Nichtfunktionale Anforderungen

ID:	NFA1
TITEL:	Entwicklungssprache und -Umgebung
BES:	Die Anwendung wird in Java mit der Entwicklungsumgebung „Android Studio“ entwickelt und soll auf Geräten mit Betriebssystemen ab Android 4.1 funktionieren.

3 Anforderungsanalyse

ID:	NFA2
TITEL:	Reaktionszeit
BES:	Die Reaktionszeit der Anwendung soll zu jeder Zeit maximal 1,5 Sekunden betragen
ID:	NFA3
TITEL:	Benutzbarkeit
BES:	Die Anwendung soll intuitiv bedienbar sein. D.h., die Buttons und andere Auswahlmöglichkeiten sollen eine ausreichende Größe haben und wie erwartet reagieren.
ID:	NFA4
TITEL:	Offline-Modus
BES:	Die Anwendung soll alle Anfragen zwischenspeichern, um auch Offline Zugriff auf die wichtigen Informationen zu gewährleisten (Bilder werden von der Bild-Bibliothek gecached).

4

Konzept, Entwurf und Implementierung

4.1 Implementierungsdetails

Wir erklären in diesem Kapitel die einzelnen Funktionen unserer Anwendung ausführlicher. Dies wird unterstützt durch Screenshots und den zugehörigen Mockups, die zu Beginn des Projekts erstellt wurden, um dem Kunden seine Vorstellungen visuell präsentieren zu können.

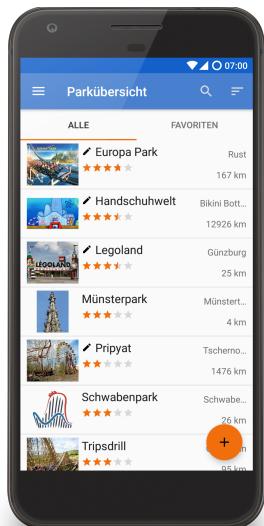


Abb. 4.1: Parkübersicht



Abb. 4.2: Mockup Parkübersicht

Nach dem Öffnen der Anwendung befindet man sich auf der Startseite unserer App. Diese ist auch zugleich die Parkübersicht, welche in Abbildung 4.1 zu sehen ist. Die Parkübersicht ist eine Liste aller Parks, die man durchsuchen und nach Alphabet, Bewertung oder Entfernung sortieren kann. Außerdem ist die Liste in zwei Tabs unterteilt.

4 Konzept, Entwurf und Implementierung

Zum Einen das Tab 'Alle', in dem, wie der Name schon sagt, alle Parks angezeigt werden, und als zweites noch das Tab 'Favoriten' in dem alle Parks angezeigt werden, die der Nutzer sich als Favorit markiert hat.

TODO Bild von der grünen Menüleiste in der man sich einloggen kann

Ist ein Nutzer eingeloggt kann er Park und Attraktionen erstellen und bearbeiten, Wartezeiten eintragen und Bewertungen abgeben. Möchte man sich nicht einloggen kann man die App trotzdem als Informationsquelle nutzen, denn man kann alle Infos wie Wartezeiten und Bewertungen sehen, aber eben nicht selber eintragen.

Falls man als Nutzer, über den orangenen Plus-Button unten rechts in der Parkübersicht, einen Park erstellt hat, wird neben dem Namen des Parks das Piktogramm eines Stifts angezeigt, was bedeutet, dass man selber Admin des Parks ist und diesen und seine Attraktionen bearbeiten kann.



Abb. 4.3: Parkansicht



Abb. 4.4: Mockup Parkansicht

Wählt man nun, durch das Tippen auf einen Park, diesen aus, so kommt man als nächstes auf die Parkdetail Seite, welche in Abbildung 4.3 zu sehen ist.

Oben auf der Seite ist immer ein Bild des Parks zu sehen. Darunter befinden sich das Herz als Symbol für 'Favorit' das sich aktivieren und deaktivieren lässt, der Parkname,

4.1 Implementierungsdetails

der Ort und falls GPS aktiviert ist auch die aktuelle Entfernung zum Park. Der Standort des Parks lässt sich außerdem durch den Button auf der rechten Seite auf GoogleMaps anzeigen.

Es befindet sich des Weiteren auch noch eine Anzeige der durchschnittlichen Bewertung (Skala von 0 bis 5) auf dieser Seite. Der Durchschnitt wird einmal als Sterne dargestellt und einmal als Dezimalzahl mit einer Nachkommastelle. Durch das Tippen auf die Sterne oder die Zahl lässt sich die Seite der Bewertungen öffnen, was aber weiter unten gezeigt wird.

Zusätzlich sind auch noch allgemeine Infos zum Park gegeben, wie z. B. die Öffnungszeiten.

Durch den am unteren Bildrand gelegenen Button 'Attraktionen' kommt man dann weiter zu den Attraktionen des Parks, die wieder, wie auch schon die Parks, als Liste gegeben sind.

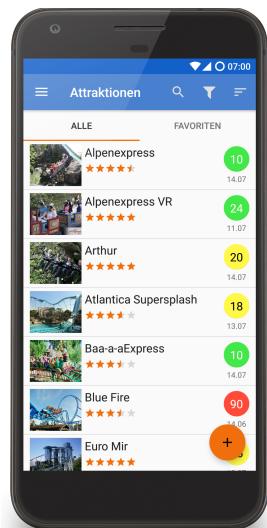


Abb. 4.5: Attraktionsübersicht



Abb. 4.6: Mockup Attraktionsübersicht

Die Seite aller Attraktionen eines Parks, die sog. Attraktionsübersicht, die in Abbildung 4.5 zu sehen ist, ist sehr ähnlich aufgebaut wie die Parkübersicht. Es gibt ebenfalls die Möglichkeiten zur Ansicht aller Attraktionen oder nur der Favoriten und zur Sortierung und Suche, aber hinzu kommt noch die Möglichkeit zum Filtern der Liste nach Kategorien der Attraktionen, wie z. B. Achterbahn oder Essensstand. Die einzelnen Elemente der

4 Konzept, Entwurf und Implementierung

Liste bestehen aus Bild, Name und Bewertung und dazuhin noch der Wartezeit in Minuten und wann diese eingetragen wurde. Die Farben der Wartezeiten bedeuten einfach dass man entweder nur kurz anstehen muss (grün), durchschnittlich lang (gelb) oder dass momentan die Wartezeit sehr hoch ist (rot). Später wird aber noch genauer auf die Wartezeiten eingegangen und auch der Algorithmus dahinter erklärt.



Abb. 4.7: Attraktionsdetail

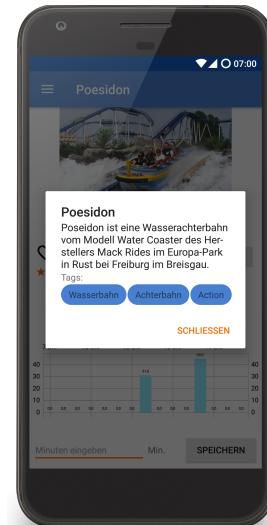


Abb. 4.8: Attraktions-Info

Wählt man nun eine Attraktion aus der Liste aus kommt man zur Detailansicht dieser Attraktion, wie in Abbildung 4.7 zu sehen.

Auf dieser Seite kann man sich, gleich wie bei Parks, die Attraktion als Favorit hinzufügen, den Standort anzeigen lassen und die Bewertungen ansehen. Neu hinzu kommt, dass man sich noch weitere Informationen über einen 'i'-Button anzeigen lassen kann, wo auch unter Anderem zu sehen ist, in welche der Kategorien die Attraktion gehört (siehe Abbildung 4.8).

Des Weiteren werden hier noch mehr Wartezeiten angezeigt, nämlich links der Durchschnitt der letzten drei Eintragungen, in der Mitte der Tagesdurchschnitt und auf der rechten Seite der Gesamtdurchschnitt. Darunter wird eine andere Wartezeitstatistik angezeigt. Hier sind auf der x-Achse die Stunden dargestellt und auf der y-Achse die Minuten. Die Säulen des Diagramms sind dann also die durchschnittliche Wartezeit in

4.1 Implementierungsdetails



Abb. 4.9: Mockup Attraktionsdetail

der angegebenen Stunde, die aus allen bisher eingetragenen Wartezeiten berechnet wird.

Ganz unten wird, falls der Nutzer eingeloggt ist, noch ein Feld zum selbst Wartezeiten eintragen angezeigt. Das Eintragen ist aber nur möglich, falls sich der Nutzer zusätzlich noch im Park befindet (Standort muss < 2km vom Park entfernt sein) und nicht schon innerhalb der letzten Stunde für diese Attraktion eine Wartezeit eingetragen hat.

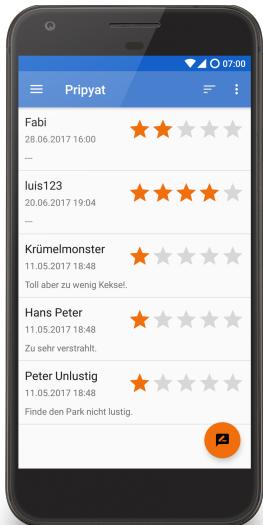


Abb. 4.10: Bewertungen



Abb. 4.11: Mockup Bewertungen

4 Konzept, Entwurf und Implementierung

Will man nun die Bewertungen einer Attraktion bzw. eines Parks sehen oder diese auch selbst bewerten, kommt man auf die in Abbildung 4.10 zu sehende Activity.

Ganz oben wird noch einmal der Durchschnitt aller Bewertungen angezeigt. Darunter sind alle bisherigen Bewertungen mit Nutzernamen, Datum und Kommentar zu sehen.

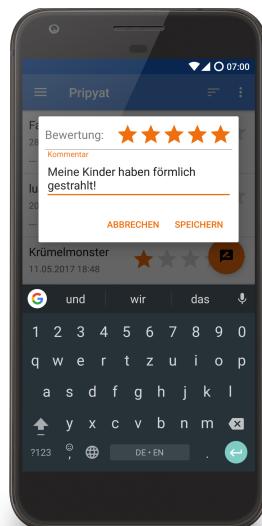


Abb. 4.12: Bewertung verfasssen



Abb. 4.13: Mockup Bewertung verfassen

Um selbst eine Bewertung abzugeben tippt man auf den orangenen Plus-Button unten Rechts und es öffnet sich ein Eingabedialog, wie in Abbildung 4.12 zu sehen. Falls man zu diesem Park bzw. dieser Attraktion eine Bewertung abgegeben hat, so wird diese hier angezeigt und man kann sie bearbeiten, aber es wird keine neue erstellt. Hat man bisher noch nicht bewertet, vergibt man jetzt 0 bis 5 Sterne, schreibt einen Kommentar und speichert dann mit 'OK'.

Für Attraktionen gibt es eine separate einfache Wartezeiten-Ansicht (vgl. Abbildung 4.12), in der die eingetragenen Wartezeiten aller Benutzer zu sehen sind. Diese lässt sich nach Name, Wartezeit, oder Datum sortieren.

Die Anwendung enthält auch einen Editor für Parks und Attraktionen, der erlaubt diese zu bearbeiten. Der Editor wird auch benutzt, um Parks und Attraktionen zu erstellen. Hier wussten wir in der Planphase noch nicht genau, was am Ende herauskommt, und so gibt es einige Abweichungen zwischen Mockup und Implementierung (vgl. Abbildung 4.16).

4.1 Implementierungsdetails

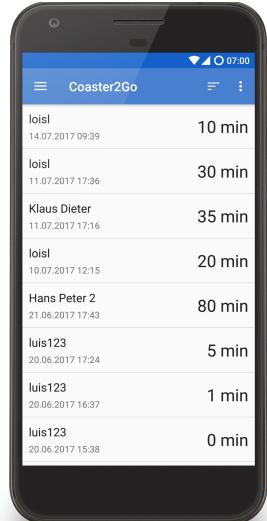


Abb. 4.14: Wartezeiten

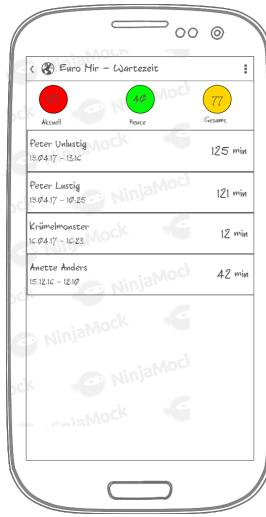


Abb. 4.15: Mockup Wartezeiten

Screenshot Todos:

- Mapview (Screenshot ist schon im Ordner)
- Menu links (Screenshot ist schon im Ordner)

4 Konzept, Entwurf und Implementierung



Abb. 4.16: Editor (Attraktion)

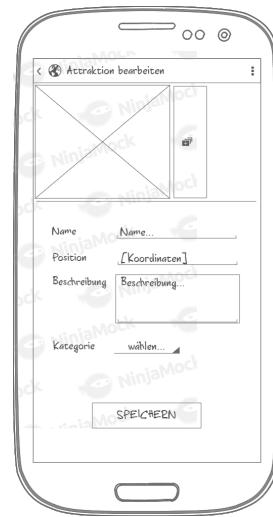


Abb. 4.17: Mockup Editor

4.2 Besonderheiten

4.2.1 Wartezeit berechnung und -darstellung

Für einen möglichst guten Überblick über die Wartezeiten einer Attraktion gibt es in unserer App verschiedene Varianten, wie der jeweilige Durchschnitt und die aktuelle Wartezeit angezeigt wird. Die verschiedenen Arten von Wartezeiten haben wir bereits in Kapitel 2.1 erklärt. In diesem Abschnitt möchten wir noch einmal kurz auf die Art der Berechnung der verschiedenen Wartezeiten eingehen.

Wir beginnen mit der Erklärung der Wartezeiten, wie sie in der Attraktionsdetail-Ansicht auf Abbildung 4.7 zu sehen sind. Am einfachsten ist sicherlich der Gesamtdurchschnitt der Wartezeiten einer Attraktion. Dieser ist ganz einfach der Durchschnitt aller eingetragenen Wartezeiten seit die Attraktion erstellt wurde, ohne irgendwelche Einschränkungen. Auch der Tagesdurchschnitt ist nicht besonders kompliziert. Er zeigt den Durchschnitt aller bereits am aktuellen Tag eingetragenen Wartezeiten für die Attraktion an. Falls am aktuellen Tag noch keine Wartezeit für eine Attraktion eingetragen wurde, wird jeweils der Tagesdurchschnitt für den letzten Tag angezeigt, an dem eine Wartezeit eingetragen wurde. Zudem wird der Tag der letzten Aktualisierung mit angegeben. Die aktuelle War-

4.2 Besonderheiten

tezeit wird derzeit als Durchschnitt der letzten drei eingetragenen Wartezeiten berechnet. Der Grund hierfür ist, dass einer einzelnen Wartezeit nicht zu viel Gewichtung gegeben werden soll, um so die Auswirkungen einer möglichen falsch eingetragenen Wartezeit zu verringern.

Für einen schnellen Überblick sind die Wartezeiten auch durch Farben gekennzeichnet. Dabei entspricht die Farbe grün, dass wenig los ist, gelb dass mittel viel los ist und rot dass viel los ist. Alle drei Wartezeiten auf der Attraktionsdetail-Ansicht sind grün, wenn sie jeweils unter 10 Minuten liegen und rot, falls sie über 90 Minuten liegen. Zudem gilt für die Aktuelle Wartezeit und für den Tagesdurchschnitt, dass sie grün sind, falls sie weniger als 70 Prozent des Gesamtdurchschnitts entsprechen und rot, falls sie mehr als 130 Prozent des Gesamtdurchschnitts betragen. In allen anderen Fällen sind sie gelb.

Die letzte Ansicht von Wartezeiten in der Attraktionsdetail-Ansicht ist ein Säulendiagramm, welches die durchschnittlichen Wartezeiten je nach Uhrzeit angibt. Dabei wird für alle Uhrzeiten zwischen 8 Uhr morgens und 20 Uhr der Durchschnitt aller Wartezeiten berechnet. Die anderen Uhrzeiten werden außen vor gelassen, da zu diesen Uhrzeiten kaum Freizeitparks geöffnet haben. Eine Wartezeit, welche zum Beispiel um 11:30 Uhr eingetragen wird, fällt somit in die Stunde zwischen 11 Uhr und 12 Uhr und wird im Diagramm in die Berechnung für die Säule "11" miteinbezogen.

In der Attraktionsübersicht, wie sie in Abbildung 4.5 zu sehen ist, wollen wir dem Nutzer einen schnellen Überblick über die Attraktionen und insbesondere über die aktuelle Wartezeit verschaffen. Um eine möglichst gute Live-Statistik der Wartezeiten mit den aktuellsten Wartezeiten anzubieten, gibt es hier eine leicht abgewandelte Berechnungsmethode. Im allgemeinen wird auch hier der Durchschnitt der letzten drei eingetragenen Wartezeiten verwendet. Gibt es aber an einem Tag bisher weniger als drei Wartezeiten, so wird nur der Durchschnitt dieser angezeigt. Sollte an einem Tag noch gar keine Wartezeit eingetragen sein, dann wird der Tagesdurchschnitt des letzten aktualisierten Tages angezeigt. Die Wartezeiten sind auch hier wieder, wie oben erklärt, farblich gekennzeichnet und zudem wird die Uhrzeit oder der Tag der letzten Aktualisierung mit angegeben. Sollte eines Tages eine Einbindung von Live-Wartezeiten über eine

4 Konzept, Entwurf und Implementierung

fremde API eines Freizeitparks geschehen, wäre diese Liste der Ort, an welchem sie eingebunden werden könnten.

Generell werden die Wartezeiten in der App von den Benutzern eingetragen. Damit ein Nutzer die Wartezeiten eintragen kann, müssen zunächst drei Bedingungen erfüllt sein:

1. Der Nutzer muss mit seinem Account eingeloggt sein. Nur so kann sicher gestellt werden, dass die Funktion nicht missbraucht wird oder ein Nutzer sehr viele Wartezeiten hoch lädt und somit den Durchschnitt manipuliert.
2. Der Nutzer muss sein GPS eingeschaltet haben und sich in einem bestimmten Umkreis zum Park befinden. Das stellt sicher, dass auch wirklich nur Nutzer Wartezeiten eintragen können, die sich auch wirklich im Park befinden.
3. Der Nutzer darf in der letzten Stunde keine Wartezeit eingetragen haben. So soll sicher gestellt werden, dass nicht ein Nutzer die Funktion missbraucht und viele Wartezeiten hoch lädt. Zudem kann angenommen werden, dass sich die Wartezeit nicht ständig ändert und ein stündlicher Intervall für das Eintragen genügt.

Hat ein Nutzer alle Bedingungen erfüllt, wird seine neue Wartezeit eingetragen. Damit nicht jedes Mal erneut alle Durchschnitte berechnet werden müssen, wenn eine Attraktion aufgerufen wird, werden alle oben genannten Durchschnitte sofort nach dem Eintragen berechnet und abgespeichert. So müssen jeweils nur noch die Werte ausgelernt werden. Damit die Berechnung nicht unnötige Rechenleistung auf dem Server verbraucht, passiert diese bereits beim Client und die neu berechneten Werte werden dann auf den Server hochgeladen. Bekanntlich kann diese Art der Aktualisierung möglicherweise zu der Lost-Update-Anomalie führen, falls zwei Nutzer zu dem exakt selben Zeitpunkt eine Wartezeit eintragen. Dies hat aber in diesem Fall keine negative Auswirkung, da ja zum selben Zeitpunkt auch die eingetragene Wartezeit die selbe sein sollte.

4.2.2 Client-Server-Kommunikation mit Azure

Für die Speicherung der Daten haben wir uns entschieden, den Dienst Microsoft Azure auszuprobieren. Dieser stellt verschiedene hilfreiche Dienste zur Verfügung, welche wir

4.2 Besonderheiten

mit einem Studentenaccount kostenlos nutzen können. Zum einen wird ein Server für die Speicherung von Daten zur Verfügung gestellt. Und dazu kommt auch gleich noch eine darauf laufende Datenbank, in unserem Fall SQLite mit node.js, auf der alle primitiven Datentypen gespeichert werden und auch online im Portal eingesehen werden können. Des Weiteren kann direkt für Mobile-Projekte auch auf eine kompakte Version von den Diensten namens Microsoft Azure Mobile Services zurückgegriffen werden. Auch eine verschlüsselte Verbindung ist mit Microsoft Azure gewährleistet.

Für unser Projekt brauchen wir nur einen geringen Teil aller Dienste, die Microsoft Azure anbietet. Wir verwenden es ausschließlich zum persistenten Speichern von Daten in einer Datenbank sowie zum schnellen Auslesen und Eintragen. Das haben wir auch nochmal in einem Modell in Abbildung 4.18 zusammengefasst.

Der Service von Microsoft bietet viele Vorteile. Zum einen ist da natürlich die Einfachheit und die gute Kontrolle über die Verwaltung der Daten durch das Online-Portal zu nennen. Auch das Zusammentreffen vieler Dienste kann als Vorteil gesehen werden. Mögliche Nachteile sind, dass manchmal die Verbindung zum Azure Server sehr langsam ist und dass man gerade als Student sehr viele Begrenzungen der Dienste in Anspruch nehmen muss.

Gerade aus dem Grund, dass die Verbindung zu Azure eine gute Internetverbindung voraussetzt und wir auch generell in unseren Nichtfunktionalen Anforderungen definiert haben, dass die Anwendung möglichst auch offline genutzt werden soll, speichern wir die wichtigsten Daten für die Anwendung zwischen. Genauer gesagt werden die Freizeitparks, und die dazu gehörigen Attraktionen jedesmal im Hintergrund zwischen gespeichert, wenn ein Nutzer diese runter lädt. Somit können auch alle relevanten Informationen zu den bereits angesehenen Freizeitparks und Attraktionen inklusiver der Wartezeiten auch ohne Internetverbindung angesehen werden. Falls keine gute Internetverbindung besteht oder die Verbindung zu lange dauert, können somit auch die Offline-Daten angezeigt werden, bis die aktuellen Daten herunter geladen wurden, damit der Nutzer nicht vor einem leeren Bildschirm auf die Daten warten muss. Um den Nutzer zudem vor unnötigen Datenabfragen vom Server zu schützen, werden die Daten auch bei wiederholtem Abfragen innerhalb kürzerer Zeiträume direkt offline geladen und

4 Konzept, Entwurf und Implementierung

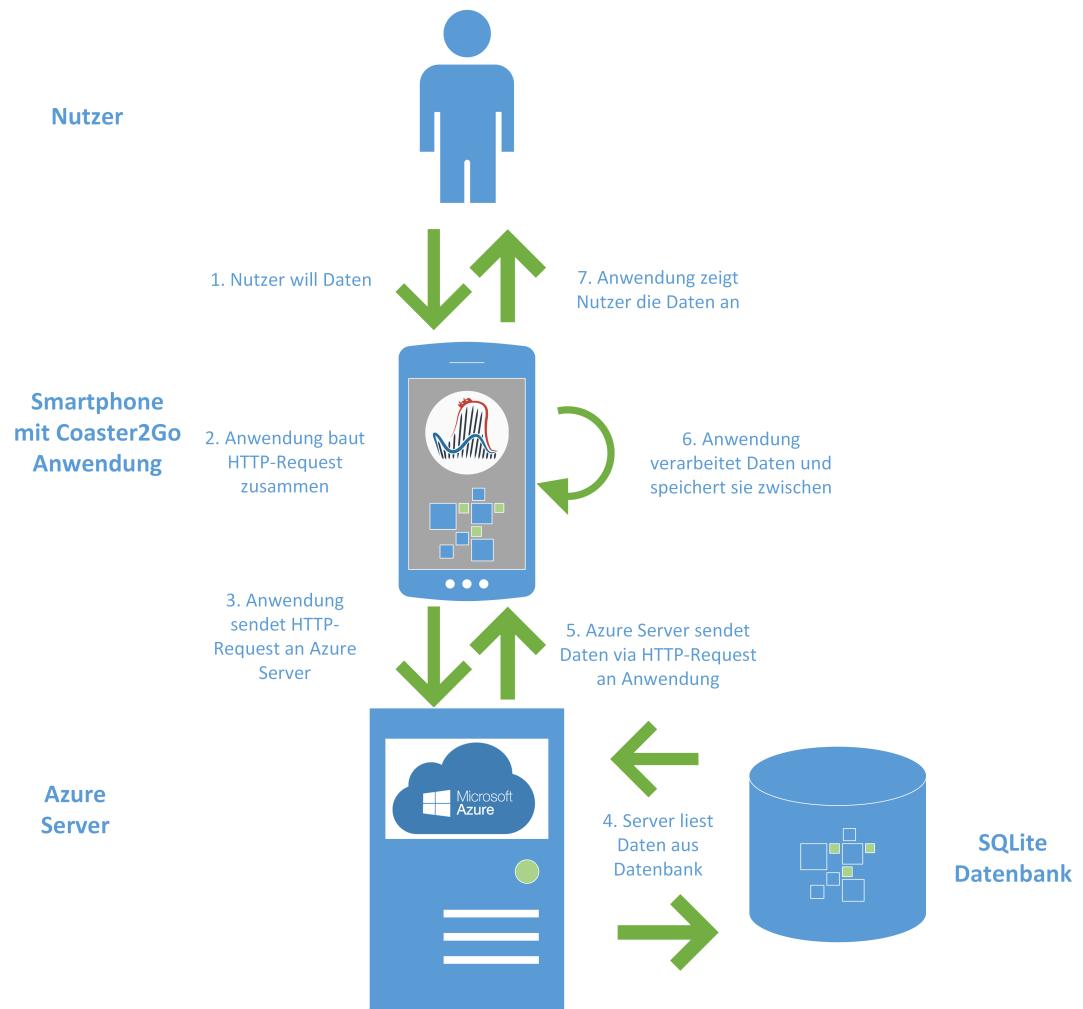


Abb. 4.18: Azure Daten Download Modell

gar nicht erst eine Anfrage an den Server geschickt. Nicht offline gespeichert werden im Gegensatz dazu Daten, die nicht zwangsweise für die Funktion der App notwendig sind, wie zum Beispiel die Liste von einzelnen Wartezeiten und Bewertungen. Bilder werden nicht offline gespeichert aber gecached, so können sie auch offline angezeigt werden und müssen nicht jedesmal erneut herunter geladen werden, solange das Betriebssystem den Speicherplatz nicht für etwas anderes benötigt.

4.2.3 Nutzerverwaltung mit Firebase

Generell wird für das Benutzen unserer App kein Nutzeraccount benötigt. Jegliche Informationen können auch ohne einen eigenen Account eingesehen werden. Bisher wird ein Account nur für zwei Aktionen benötigt: das Eintragen von Bewertungen und das Eintragen von Wartezeiten. Bei beidem soll der Account zum einen vor Missbrauch schützen und zum anderen die Nutzer eindeutig identifizieren. Hinzu kommen noch die Adminfunktionen. Um Parks verwalten zu können, kann ein Nutzer Administrator von einem Park werden. Dieser Nutzer hat dann einige weitere Funktionen, welche den Park betreffen, wie das Bearbeiten der Funktionen und Bilder des Parks, das Hinzufügen und Bearbeiten von Attraktionen zu diesem Park und das Zensieren von anstößigen Bewertungstexten. Während der Testphase der App kann jeder Nutzer einen eigenen Park erstellen und Administrator davon werden.

Da wir eine sehr einfache Nutzerverwaltung haben und neben einem Displaynamen nur die Email und ein Passwort benötigen, wäre es keine Schwierigkeit gewesen, eine eigene Nutzerverwaltung zu schreiben. Der Grund, warum wir dennoch auf einen Fremdienst für die Nutzerverwaltung zurückgegriffen haben ist, dass wir dem Nutzer den Aufwand erleichtern wollen und ihm auch eine Anmeldung mit einem bereits vorhandenem Konto eines Social Media Dienstes ermöglichen wollen. Da es ein großer Aufwand wäre, diese für jeden Dienst einzeln einzubinden, haben wir uns dafür entschieden, die Nutzerverwaltung mit einem Dienst zu realisieren, welcher viele verschiedene Social Media Accounts gemeinsam verwalten kann. In unserem Fall fiel die Wahl auf Firebase.

Firebase, welches mittlerweile zu Google gehört, bietet eine Reihe an Funktionen wie Datenverwaltung, Datensynchronisation oder Nutzerverwaltungen. Für uns war lediglich

4 Konzept, Entwurf und Implementierung

letzteres relevant. Durch die Einbindung von Firebase in unsere App ist es den Nutzern nun möglich, sich direkt mit ihrem Facebook oder Google Account einzuloggen und die App zu benutzen. Nach der Authentifizierung über den Fremddienst verknüpft Firebase automatisch einen Account unserer App mit dem jeweiligen Account des Fremddienstes. Diesen Vorgang haben wir auch noch einmal in einem Modell in Abbildung 4.19 zusammengefasst. Des Weiteren kann natürlich auch ein neuer Account über eine Emailadresse angelegt werden. Ist ein Nutzer eingeloggt, bleibt er so lange auch nach Beenden der App eingeloggt, bis er sich wieder ausloggt und sich dann erneut direkt mit seinem Account einloggen kann. Um die Privatsphäre einer Person zu schützen, ist es zudem möglich, den Namen, welcher den anderen Nutzern angezeigt wird, zu ändern, damit zum Beispiel nicht der echte Name verwendet wird.

Firebase bietet den Vorteil, dass es einem sämtliche Arbeiten im Bereich der Nutzerverwaltung abnimmt und man sich selbst nicht mehr groß damit beschäftigen muss. Der größte Vorteil ist auf jeden Fall die Einbindung der vielen anderen Social Media Dienste. Ein Nachteil sind die oft langen Ladezeiten beim Einloggen sowie ein reisen Overhead an benötigten Bibliotheken und Ressourcen für eine eigentlich nicht aufwendige Aufgabe.

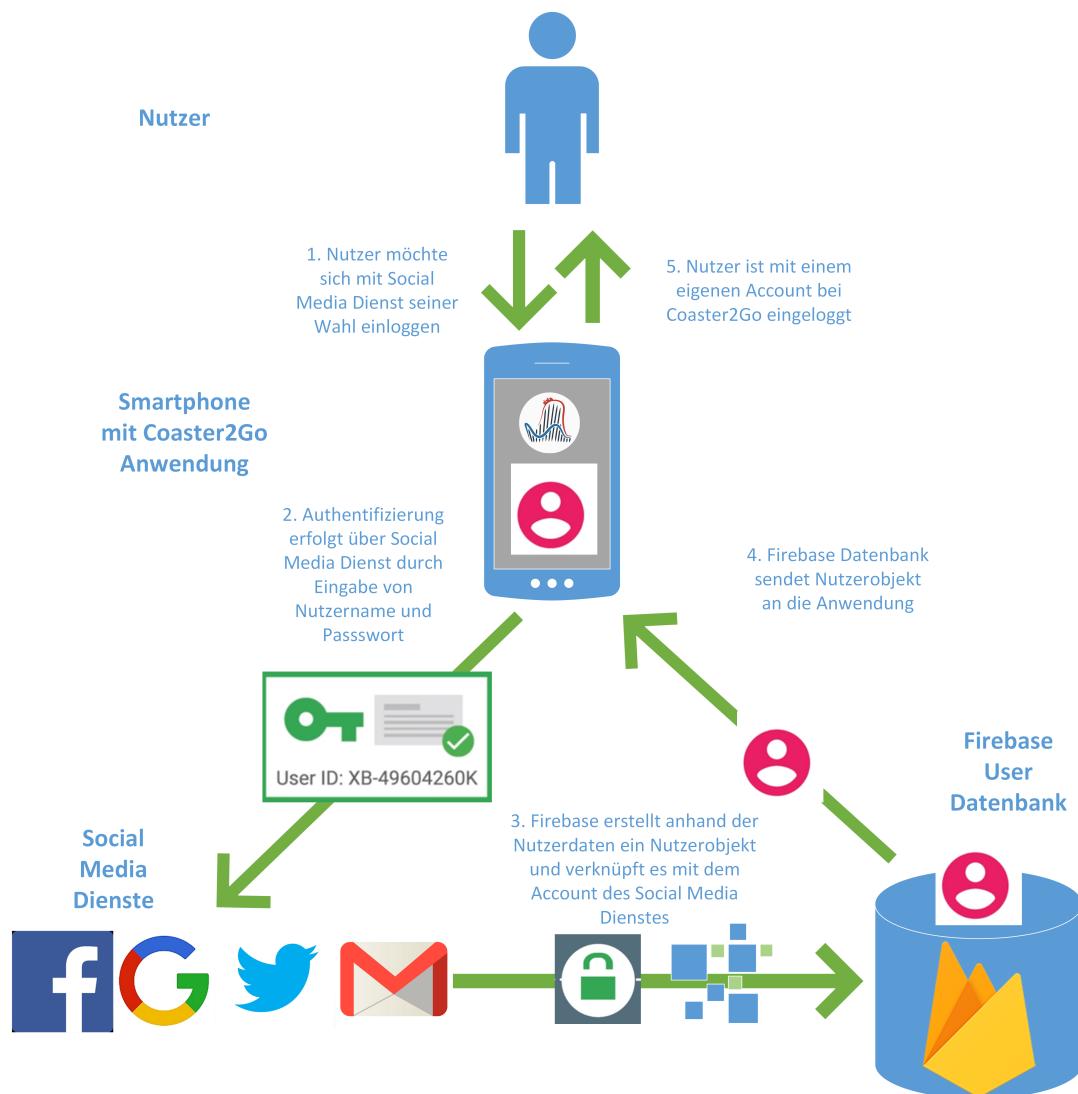


Abb. 4.19: Firebase Nutzerverwaltung Modell

4.3 Architektur

4.3.1 Datenmodell und -speicherung

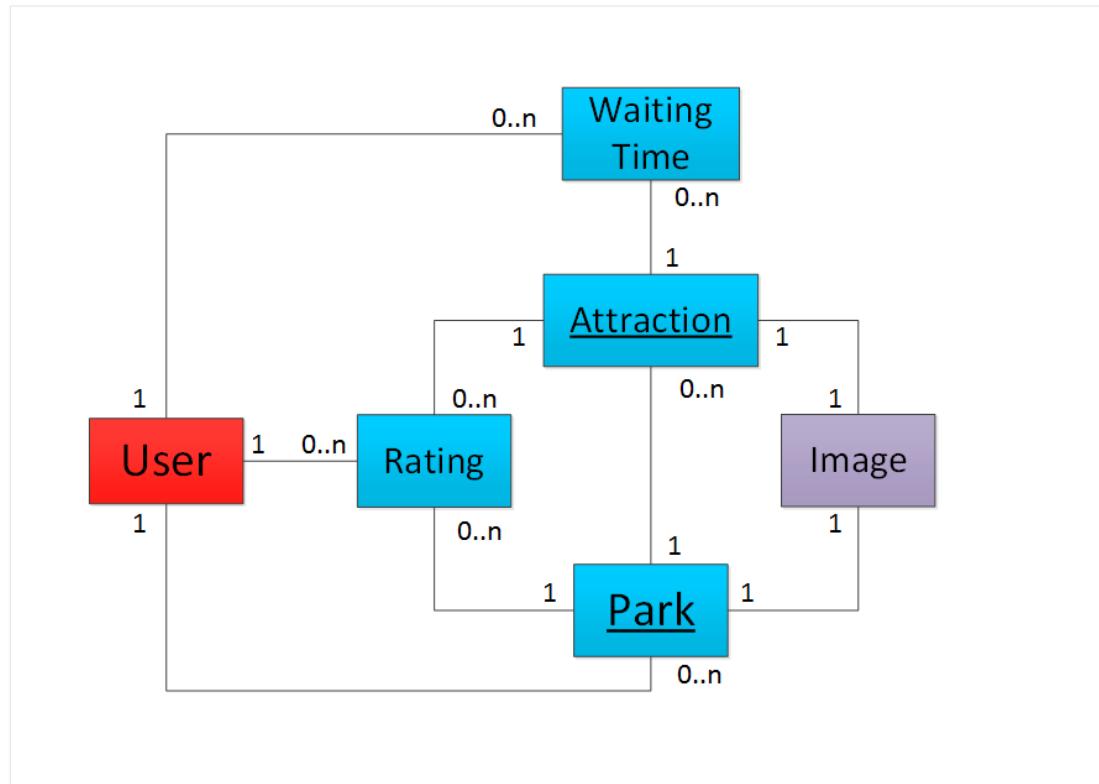


Abb. 4.20: Datenmodell

In Abbildung 4.20 ist das Datenmodell für unsere App zu sehen. Es ist in diesem Fall nicht besonders kompliziert. Deswegen haben wir auch auf die Darstellung der einzelnen Attribute verzichtet, da diese sich implizit aus der Anwendungslogik ergeben. Die verschiedenen Farben sollen den Speicherort der Daten unterscheiden. Alle blauen Entitäten werden in Form von Tabellen in einer SQLite Datenbank auf einem Microsoft Azure Server gespeichert. Die User-Daten werden von Firebase auf einem Firebase Server gespeichert. Und die Bilder werden über Cloudinary gespeichert. Der Grund hierfür ist zum einen, dass wir mit einem Studentenaccount keine Dateien bei Azure speichern können, und zum anderen, dass über Cloudinary Bilder ganz einfach über einen Http-Request geladen und gleich auch noch transformiert werden können. Die

Daten der beiden unterstrichenen Objekte (Attraction und Park) werden zusätzlich auch auf dem Handy in Form von JSON-Dateien gespeichert.

4.3.2 Klassen- und Activity-Architektur

In Abbildung 4.21 sieht man das Klassenmodell bzw. das Activity-Modell unserer App. Dort sind die einzelnen Activities und damit gleichzeitig Views unserer App und deren Verknüpfungen dargestellt. Die unterschiedlichen Farben sollen die Art der Activities symbolisieren. Die BaseActivity ist die Grund-Activity, von der alle anderen Activities erben, und auf die jederzeit zugegriffen werden kann. Die ParkOverviewActivity ist die Activity, die der Nutzer sieht, wenn er die App startet. Zusätzlich zu den Activities gibt es noch die Datenklassen, Klassen für die Onlinekommunikation und Offlinespeicherung der Daten, sowie einige Helferklassen und Klassen für Listadapter.

4.3.3 Arbeitsaufteilung

Während der Implementation wurden die Aufgaben so aufgeteilt, dass jeder aus dem Team immer für eine bestimmte Aufgabe zuständig war und diese durch alle Schichten durch erledigen musste. Bei großen Aufgaben, wie den Übersichten oder den Detailansichten, wurden die Aufgaben zudem in kleinere Schritte aufgeteilt. Da die App aus mehreren unabhängigen Bereichen besteht, konnten die Aufgaben gut eingeteilt werden, ohne dass es zu Behinderungen kommt. Wenn gerade keine Arbeit für eine Person angefallen war, beschäftigte diese sich mit Verbesserungen, Testen von Funktionen oder mit der Weiterentwicklung des Designs.

4 Konzept, Entwurf und Implementierung



Abb. 4.21: Klassenmodell

4.4 Herausforderungen während der Implementierung

In diesem Abschnitt möchten wir noch auf einige Herausforderungen und Probleme eingehen, die wir während der Implementierung hatten, und wie wir diese gelöst haben. Generell muss man dazu sagen, dass es insgesamt doch zu deutlich weniger Problemen gekommen ist, als wir anfangs vermutet hatten. Und wenn Probleme aufkamen, hat man dank der ausführlichen Dokumentation und dank vielen Hilfen im Internet relativ schnell die Lösung dafür gefunden.

- Unterstützung älterer Versionen bei Benutzung neuer Funktionen: Für fast alles gibt es bei Android mittlerweile bereits implementierte Funktionen, jedoch sind viele dieser Funktionen erst ab einer bestimmten Version verfügbar. Da wir unsere App aber explizit auch für ältere Android Modelle kompatibel machen wollten, mussten wir öfter die einfachere Variante außen vor lassen und etwas selbst implementieren.
- Dex-Limit bei Android oder auch genannt das 64k Methoden-Limit: Dieses Limit taucht auf, wenn die App insgesamt mit allen Bibliotheken über 64.000 Methoden besitzt und auch mit älteren Android-Versionen kompatibel sein soll. Die vielen Methoden kommen dadurch zustande, wenn man mehrere Bibliotheken einbindet, die selbst wiederum abhängig zu anderen Bibliotheken sind. So hat zum Beispiel alleine die Google-Play-Bibliothek über die Hälfte der Methoden ausgefüllt. Oft ist es so, dass nur eine Funktion einer Bibliothek gebraucht wird, sie aber komplett eingebunden werden muss. So haben wir trotz nicht all zu vieler Einbindungen recht schnell dieses Limit überschritten. Mit ein wenig Recherche konnte aber auch dieses Problem umgangen werden.
- Interface-Design so umsetzen wie geplant: Wie auch schon im letzten Semester kann man festhalten, dass das Designen der GUI mit Android Studio sehr angenehm ist. Trotzdem hat es uns erneut vor eine Herausforderung gestellt, die GUI auch so umzustzen wie geplant. Es gab alleine zwischen unseren Handys und Emulatoren öfter mal größere Unterschiede bei der Darstellung.

4 Konzept, Entwurf und Implementierung

- Microsoft Azure: Azure bietet zwar viele Vorteile aber auch einige Schwierigkeiten, wie ein nicht intuitives Webinterface, Einschränkungen für Studenten und einen scheinbar zwischenzeitlich zufälligen sehr langsamen Verbindungsauftbau.
- Möglichst wenige Verbindungen zum Server aufzubauen: Die Daten schlau herunter zu laden und zwischen zu speichern, so dass der Client möglichst wenig Anfragen an den Server schicken muss, war nicht immer selbstverständlich. Es muss oft abgeschätzt werden, wann eine Anfrage an den Server sinnvoll ist und wann auf zwischengespeicherte Daten zurück gegriffen werden kann.
- Benutzung von Google-Services: Das Arbeiten mit bereits vorhandenen APIs wie zum Beispiel den Maps- / Standort-Services musste erst erlernt werden. Aber dank der guten Dokumentation war dies kein Problem.

5

Anforderungsabgleich

Es folgt ein tabellarischer Vergleich zwischen den in Kapitel 3 spezifizierten Anforderungen und den implementierten Funktionen. Es wird für jede der Anforderungen beurteilt, inwieweit diese in der Implementierung erfüllt ist (Ja / Teilweise / Nein). Eine Zuordnung zu den Einträgen aus Kapitel 3 ist anhand der ID möglich.

Diejenigen Funktionen, die implementiert wurden, jedoch nicht im Voraus als Anforderungen spezifiziert waren, werden in diesem Vergleich nicht berücksichtigt.

5 Anforderungsabgleich

5.1 Funktionale Anforderungen

ID	Erfüllt	Kommentar
FA1	Nein	Unnötig, da die Account-Funktionen im Sidebar-Menü untergebracht wurden und die Anwendung direkt auf der Parkübersicht startet.
FA2	Ja	-
FA3	Ja	-
FA4	Ja	-
FA5	Ja	-
FA6	Ja	-
FA7	Ja	-
FA8	Ja	Es sind weiterhin 4 verschiedene Statistiken, jedoch wurde die Semantik leicht geändert, um sie hilfreicher zu machen (siehe Kapitel 4.2.1 "Wartezeit-Berechnung und -darstellung").
FA9	Ja	-
FA10	Ja	-

5.2 Nichtfunktionale Anforderungen

ID	Erfüllt	Kommentar
NFA1	Ja	-
NFA2	Ja	Die Zeit zum Hoch- oder Herunterladen von Daten liegt oft weit über 1,5 Sekunden, aber dieser Vorgang blockiert nicht die Benutzeroberfläche und kann abgebrochen werden.
NFA3	Ja	Die Anforderung ist nur schwer zu beurteilen. Es wurde sich möglichst an die Android-Richtlinien für Benutzeroberflächen gehalten, und alle Bereiche der Anwendung sind nach dem Empfinden aller bisherigen Benutzer leicht bedienbar.
NFA4	Ja	-

6

Ausblick

Wie im Kapitel 5 zu sehen ist, haben wir all unsere inhaltlichen Anforderungen an die App erfüllt. Deswegen haben wir uns in diesem Kapitel einige Gedanken gemacht, welche Funktionen für eine zukünftige Erweiterung der App denkbar wären. Diese könnten die bisherigen Funktionen sinnvoll ergänzen.

- Wartezeitenverifizierung: Bisher gibt es in unserer App keine Möglichkeit, falsch eingetragene Wartezeiten zu melden oder rückgängig zu machen. Zwar gleichen sich einzelne falsch eingetragene Wartezeiten dank der verwendeten Metriken zur Berechnung der Durchschnittswartezeiten schnell aus und haben keinen großen Einfluss. Trotzdem wäre es eine gute Ergänzung, wenn andere Nutzer die zuletzt eingetragenen Wartezeiten bestätigen oder ablehnen könnten, sodass die jeweilige Wartezeit an Gewichtung zu- oder abnimmt.
- Ansteh-Timer: Um die Wartezeit nicht jedes mal selbst messen oder ablesen zu müssen, wäre ein Ansteh-Timer eine gute Ergänzung für die Benutzer der App. Dazu wird beim Beginn des Anstehens ein Startknopf gedrückt, welcher den Timer startet. Kurz vor der Fahrt oder direkt nach der Fahrt kann dann der Stopptknopf gedrückt werden. Dadurch wird die gewartete Zeit automatisch ermittelt und hochgehalten.
- Bewertungskategorisierung: Bisher gibt es bei unseren Bewertungen für die Parks und Attraktionen jeweils nur die Möglichkeit, allgemein zwischen null und fünf Sternen zu verteilen. Noch besser wäre, für jeden Park oder jede Attraktion bestimmte Kriterien der Kategorisierung vorzunehmen. So könnten zum Beispiel Achterbahnen nach der Action, Wasserbahnen nach ihrer Erfrischung, Familienbahnen nach

6 Ausblick

der Tauglichkeit für Kinder oder Restaurants nach dem Geschmack bewertet werden. Dies bietet sich besonders bei Attraktionen an, da sie zum einen bereits in verschiedene Kategorien aufgeteilt sind. Zum anderen laufen die Benutzer so nicht in Gefahr, zum Beispiel eine Kinderbahn niedriger zu bewerten, weil sie nicht genügend Action für sie bereit hält.

- Tourplaner: Da die Attraktionen schon in einer Mapview angezeigt werden, wäre es eine sinnvolle Ergänzung, den Benutzern die Möglichkeit zu geben, sich selbst eine Tour zusammen zu stellen, welche sie dann auf der Karte angezeigt bekommen. Des Weiteren könnten auch bereits vorhandene Touren geladen werden oder Touren automatisch erstellt werden, zum Beispiel nach Kategorien oder nach Wartezeiten.

Zudem besteht zumindest theoretisch die Möglichkeit, Freizeitparks und ihre Daten direkt mit in die App einzubinden. Dadurch ergibt sich auch die Möglichkeit, wie sich die App finanzieren könnte. Freizeitparks, die eine Kooperation eingehen, könnten Vorteile oder Premiumaccounts erhalten. Mögliche Vorteile wären neben dem selbstständigen Verwalten der eigenen Parkdaten zum Beispiel das Einbinden ihrer eigenen Live-Wartezeiten aus bereits vorhandenen parkeigenen APIs, das Schalten von Werbung oder Aktionen des Freizeitparks oder auch das Hervorheben des Freizeitparks durch ein besonderes Design oder eine gut sichtbare Platzierung in der Übersichtsliste.

7

Fazit

In diesem Kapitel soll ein kurzes Fazit zur App-Entwicklung und zum Anwendungsfach allgemein festgehalten werden.

Dadurch, dass wir als Team schon im Semester zuvor die Quartett-App zusammen entwickelt haben, hatten wir schon genügend Vorwissen, um in die Entwicklung unserer Freizeitpark-App einzusteigen. Trotzdem haben wir auch während der Entwicklung dieser App viel Neues gelernt was wir für die Zukunft und für die Entwicklung weiterer Apps gebrauchen können, wie zum Beispiel das Arbeiten mit Standorten und das Einbinden von vorhandenen Frameworks.

In unseren Augen ist uns die Coaster2Go App größtenteils gut gelungen und das Endergebnis ist genau so, wie wir es uns am Anfang des Semesters vorgestellt haben. Die App wäre theoretisch schon ausreichend für eine Veröffentlichung im App Store, da es bisher keine vergleichbare App im Google Play Store gibt. Natürlich gäbe es noch einige Stellen, die eine Anpassung benötigen, bevor es wirklich zu einer Veröffentlichung kommen kann. Zum Beispiel müsste die Speicherung der Daten auf einen anderen Server verlegt werden, da wir bisher mit unseren Studentenaccounts nur begrenzte Kapazitäten besitzen. Auch kleinere gestalttechnische oder funktionale Verbesserungen sind denkbar.

Auch was das Arbeiten im Team und die Aufteilung der Aufgaben angeht, konnten wir im Vergleich zum ersten Teil des Anwendungsfaches noch Neues dazu lernen. Die Teamarbeit hat immer gut funktioniert und es kam nie zu größeren Komplikationen. Wir werden es auf jeden Fall in Erwägung ziehen, möglicherweise im Masterstudium das Anwendungsfach fortzusetzen.

Literaturverzeichnis

Abbildungsverzeichnis

1.1	Blue Fire Achterbahn	2
1.2	Coaster2go Logo	3
2.1	Wartezeitanzeige im Park	5
2.2	Wartezeitübersicht im Europapark	6
2.3	Android Zustandsmodell	8
4.1	Parkübersicht	15
4.2	Mockup Parkübersicht	15
4.3	Parkansicht	16
4.4	Mockup Parkansicht	16
4.5	Attraktionsübersicht	17
4.6	Mockup Attraktionsübersicht	17
4.7	Attraktionsdetail	18
4.8	Attraktions-Info	18
4.9	Mockup Attraktionsdetail	19
4.10	Bewertungen	19
4.11	Mockup Bewertungen	19
4.12	Bewertung verfassen	20
4.13	Mockup Bewertung verfassen	20
4.14	Wartezeiten	21
4.15	Mockup Wartezeiten	21
4.16	Editor (Attraktion)	22
4.17	Mockup Editor	22
4.18	Azure Daten Download Modell	26
4.19	Firebase Nutzerverwaltung Modell	29
4.20	Datenmodell	30
4.21	Klassenmodell	32