HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

COMPUTER SCIENCE

Kort Native - OpenStreetMap Data Completion Through Gamification

Master's Thesis

Department of Computer Science
University of Applied Sciences Rapperswil

Spring Term 2017

Author: **Andreas Egloff**

Advisor: **Prof. Stefan Keller**

Imprint

Authors:	Andreas Egloff (egloff.andreas@gmail.com)
Creation Date:	June 15, 2017
Last Update:	July 25, 2017

This document was created using LaTeX.

Abstract

The core idea of the collaborative and free project *OpenStreetMap* (OSM) is that everyone should be able to contribute to the community by collecting data by manual survey, GPS or satellite imagery and make this data available as open data. Nowadays however, it is not sufficient for a mapping software to just present the latest geometry features. The requirements of such software have increased especially with smartphones where one expects further information on geographic features e.g opening hours of a supermarket or cuisine type of a restaurant etc.

In order to tackle this problem, a variety of expert tools for OSM exist to edit this data. However, for a much broader audience, these tools are not accesible. That is why in 2012 the game *Kort* was launched. It was a gamified web app by means of which one can solve tasks which enhanced map data. Although quite a success, the game was discontinued.

This work consists of a complete reboot of the project featuring a native solution. Some of the gamification elements have been adopted and improved. People using *Kort* can solve missions and earn so-called «Koins» in order to climb up in the leaderboards. In order to keep users coming back, a plethora of achievements has been introduced which users can obtain by solving more missions, hence improving OSM data altogether.

The implementation in this work makes use of a wide range of different technologies. The backend is written in *Python* and uses the *Connexion* framework by *Zalando* which sits on top of *Flask* and exposes a RESTful API based on OpenAPI 2.0 Specification (FKA Swagger Spec). The services as well as the PostgreSQL database run in *Docker* containers, enabling easy deployments on different systems. The frontend is written in JavaScript and uses the *React Native* framework by *Facebook*. This allows to have true native mobile apps while maintaining only one single codebase.

The result of this work is a native mobile app for the Android and the iOS platform as well as an all new backend. A new error source (Overpass) with new mission types has been added as well. The work could be successfully presented at SotM-FR and received broad interest which resulted in many beta testers. The app is currently in beta, and due to be publicly released later this summer.

Further information is available on www.kort.ch.

Keywords: OpenStreetMap, React Native, Gamification, iOS, Android

Management Summary

Problem Statement

The core idea of the collaborative and free project *OpenStreetMap* (OSM) is that everyone should be able to contribute to the community by collecting data by manual survey, GPS or satellite imagery and make this data available as open data. Nowadays however, it is not sufficient for a mapping software to just present the latest geometry features. The requirements of such software have increased especially with smartphones where one expects further information on geographic features, e.g opening hours of a supermarket or cuisine type of a restaurant etc.

In order to tackle this problem, a variety of expert tools for OSM exist to edit this data. However, for a much broader audience, these tools are not accesible.

Goals

Motivated to simplify this workflow, a Bachelor's Thesis [22] addressed this issue in 2012 which resulted in the game *Kort*, a gamified Web app to solve tasks which enhanced map data. Although quite a success, the game was discontinued. This work's main goal is to reboot the project and enhance it in many ways.

First of all, it is important to strengthen the established brand *Kort* and revive the game. Some of the gamification elements have been adopted and improved. People using *Kort* can solve missions and earn so-called «Koins» in order to climb up in the leaderboards.

Second of all, the Web app is to be abandoned in favor of Native Apps, which are much swifter to work with. Nowadays, two major mobile platform exist, Android and iOS. The main disadvantage over Web apps is naturally the bigger coding effort, since these platforms require different programming languages (Swift, Objective C and Java). From a software engineering point of view, it is always preferred to have a single codebase for as much as possible. Therefore, it was decided to develop the mobile apps with the JavaScript framework *React Native* by *Facebook*. This allows to have true native mobile apps while maintaining only one single codebase.

Third of all, the existing backend consisting of a PHP Server and a PostgreSQL database is to be replaced by a solution which is easier to maintain.

Finally, the ultimate goal is to revive the project and bring those apps to the App Store and Google Play Store.

Achievements

At the very beginning of the project it was decided not only to develop an all new frontend, but rather replace the existing backend as well. This resulted in several architectural decisions based on the requirements given.

First, in order to make the backend easier to maintain, it was clear to use some sort of containerization. That means that an application is encapsulated in a container with its own operating environment, getting rid of any dependencies as well as simplifying the deployment process. This architecture was realized with *Docker*[1]. Each component of the backend lives in its own container, e.g. PostgreSQL database and API.

Second, the process of adding new error sources (missing data) has been extended. The data gathering process now includes OSM errors not only from *KeepRight*[2], but also from *Overpass*[3]. By using the Overpass API it is now possible to easily add new mission types for a global dataset. In scope of this work, this extension has been applied for places with missing opening hours among others.

Third, the PHP API has been rewritten with easier to read *Python* code. It makes use of the *Connexion*[4] framework by *Zalando* which sits on top of *Flask* and exposes a RESTful API based on OpenAPI 2.0 Specification (FKA Swagger Spec) [4]. This allows a clear and up-to-date live documentation of the interface since Connexion works with a specification-first approach and not the other way around. As a huge bonus, the Swagger Web Console UI allows easy testing by calling the API's endpoint directly.

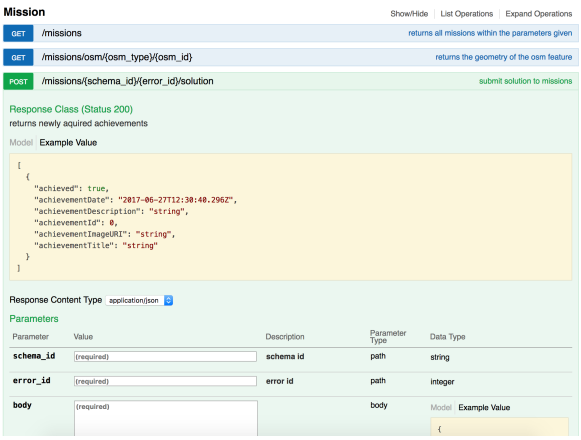

Figure 0.1.: Swagger API Documentation

[1] https://www.docker.com/
[2] http://www.keepright.at/
[3] https://overpass-turbo.eu/
[4] https://github.com/zalando/connexion

Finally and most importantly, the frontend has been redesigned and rewritten with *React Native*. Now, there exists a native app for the Android and the iOS platform. Users can login with their Google or OSM account, which was an important requirement, since users do not like to create new accounts and remember new passwords. Behind the scenes, the API uses the OAuth standard. Nevertheless, it is also possible to discover the app without logging in.

The map features vector tiles by the *OpenMapTiles*[5] project. Compared to raster tiles, this allows reducing data transfers dramatically which is essential in mobile applications.

Furthermore, new mission types have been introduced, namely missing building level, missing cuisines for restaurants and missing opening hours for community facilities. An adapting input mask is presented for each mission type, e.g. for opening hours a native date/time picker is presented. Along with new missions, the number of achievements could also be increased. Now there exist achievements for each mission type, as well as a special achievement which can be unlocked when solving six missions in one day.

In order to increase the competition with other players, more leaderboards have been introduced. It is now possible to be the leader of the day, week or month.

A showcase featuring the possibilities is presented when the player uses the app the first time.

The work could be successfully presented at SotM-FR and received broad interest which resulted in many beta testers.

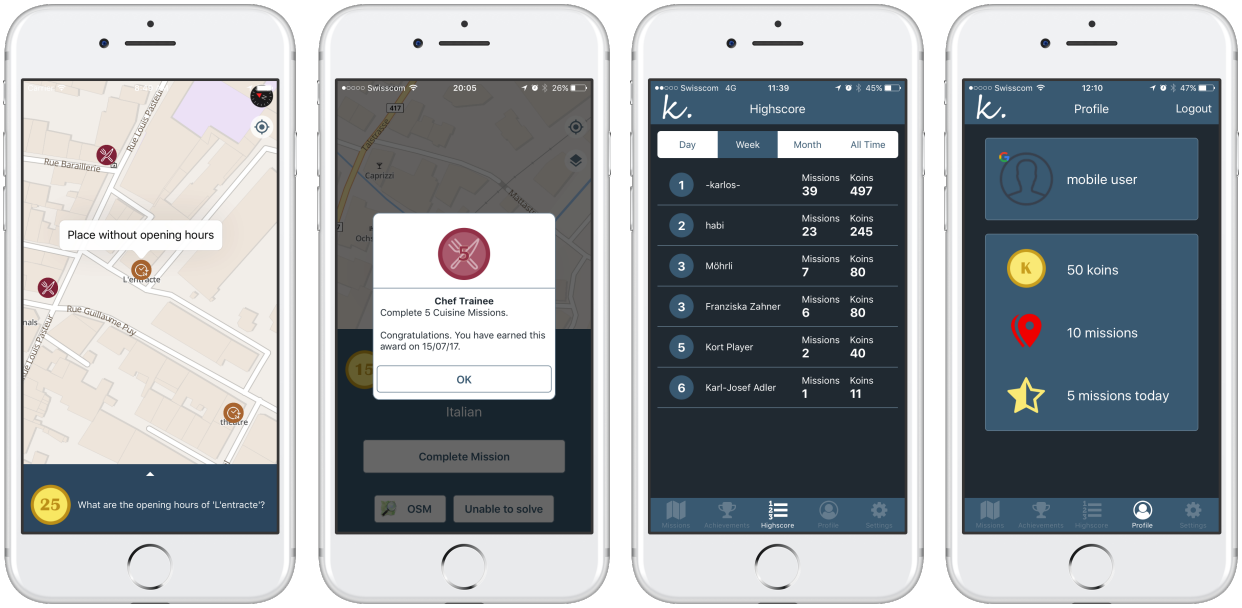

Figure 0.2.: Kort Native App

[5] https://openmaptiles.org/

Outlook

The groundwork has been laid. The next goal is to bring the app to the App Store and Google Play Store. This will be happening shortly after a successful beta phase, which is still ongoing as of this writing.

After a successful launch later this summer, it is possible to add further enhancements, some of which shall be presented at this point.

First of all, it is vital to add a module to the backend which automatically writes back validated missions to OSM. This will strengthen the reputation of this game in the OSM community. This could be achieved by creating an additional Docker container responsible for that job. On an additional note, the mission creation/update process needs to be accelerated when requesting data from the Overpass API. One could also think of having a separate API to add missions sources. In a basic variant this could look as a way of adding Overpass queries with additional attributions. Before going live with the new Kort, the old data needs to get migrated. This is important because returning players wish to keep their previous accomplishments.

Second of all, the statistics gathered with iTunesConnect, Google Play Store and Mapbox telemetry[6] should be used to further improve existing parts of the app. App Store reviews should be addressed if needed.

Third of all, it is possible to add more spatial awareness, such as spatial achievements, leaderboards or promotions. The latter was available in the old Web app and was not a goal in scope of this work.

Fourth of all, the fact that this game is now a true Native App opens up the possibilities to have push notifications (e.g. for promotions) as well as Deep Linking from and to other apps such as Facebook, Whatsapp or mapping apps. It is also possible to create offline functions, for instance the ability to download the map for a whole region and sync the solved missions when online again.

Fifth of all, when the React Mapbox GL library comes out of its infancy, there is better support on both platforms for custom map annotations. This allows to have better performance and more layout options on map annotations. Furthermore, it may be possible to do a clustering on annotations thus hinting at hot spots of missions at any zoom level at all times.

Finally, additional gamification elements could be introduced such as experience points (XP) or levels [17]. Levels do not necessarily need to be numbers, but rather funny titles. These could then also be visible in the leaderboards.

[6] https://www.mapbox.com/telemetry/

Declaration of Originality

I hereby declare that

- this thesis and the work reported herein was composed by and originated entirely from me unless stated otherwise in the assignment of tasks or agreed otherwise in writing with the supervisor.
- all information derived from the published and unpublished work of others has been acknowledged in the text and references are correctly given in the bibliography.
- no copyrighted material (e.g. images) has been used illicitly in this work.

Place, Date

Signature

Acknowledgments

I express my sense of gratitude to my supervisor Prof. Stefan Keller. Thanks to his idea of rebooting the project I was able to work with new technologies and tools, thus, broadening my mind to a great extent. His invaluable insight into the matter guided me throughout the work.

I would also like to take this opportunity to thank Sebastian Deutsch, React guru and workshop speaker. He helped me at the very beginning of the project with React, Redux and React Native, all of which was completely new to me.

I am very much thankful to Marcel Huber and Prof. Peter Sommerlad for giving me access to test infrastructure at HSR where I could run my beta tests.

I acknowledge with thanks the feedbacks from many beta testers from SotM-FR or elsewhere. It is always encouraging to get detailed reports on what can be improved.

And last, but not least, I am extremely thankful to Franziska Zahner for her constant encouragement throughout this work and during my studies altogether.

Contents

Part I.

Technical Report

1. Introduction

The technical report consists of seven main parts. The first chapter defines the problem statement as well as what goals need to be achieved. A short overview over the state of the art concludes this chapter. The seconds chapter conducts an evaluation on the technologies in terms of the requirements. The third chapter pinpoints the single parts of a typical requirements engineering process covering use cases, sequence and activity diagrams, as well as non-functional requirements. The fourth chapter sketches an architectural view on the parts to be implemented. The fifth chapter covers the design aspect in terms of interaction as well as gamification elements. The sixth chapter delves into the implementation itself. It highlights parts of the implementation which are worth mentioning. The seventh chapter ultimately presents the results that were achieved in scope of this work and gives a possible outlook for further work. It summarizes with a personal reflection on the project.

1.1. Problem Statement

The core idea of the collaborative and free project *OpenStreetMap*(OSM) is that everyone should be able to contribute to the community and make this data available as open data. This includes manual survey, GPS or satellite imagery. Nowadays however, it is not sufficient for a mapping software to just present map data. The requirements of such software have increased especially with smartphones where one expects further information on geographic features e.g opening hours of a supermarket or cuisine type of a restaurant etc.

In order to tackle this problem, a variety of expert tools for OSM exist to edit this data. However, for a much broader audience, these tools are not accesible.

1.2. Project Aim

Motivated to simplify this workflow, a Bachelor's Thesis [22] addressed this issue in 2012 which resulted in the game *Kort*. It was a gamified Web app with the help of which one could solve tasks which enhanced map data. Although quite a success, the game was discontinued. This work's main goal is to reboot the project and enhance it in many ways.

First of all, it is important to strengthen the established brand *Kort* and revive the game. The name *Kort* originally derives from the Old Norse spoken by inhabitants of Scandinavia and means «map». It is still used today in Danish, Faroeseor or Icelandic [7].

People using *Kort* can solve missions and earn so-called «Koins» in order to climb up in

the leaderboards. All of the gamification elements should stay included and improved where possible. Namely these are the following:

- Koins (Points, Rewards)
- Achievements (Badges)
- Highscore (Leaderboards)

Second of all, the Web app is to be abandoned in favor of native mobile apps, which are much swifter to work with. Nowadays, two major mobile platform exist, Android and iOS. The main disadvantage over Web apps is naturally the bigger coding effort, since these platforms allow different programming languages (Swift, Objective C and Java). From a software engineering point of view, it is always preferred to have a single codebase for as much as possible. Therefore, it was decided to develop the mobile apps with the JavaScript Framework *React Native* by *Facebook*. This allows to have true native mobile apps while maintaining only one single codebase.

Third of all, the existing backend consisting of a PHP Server and a PostgreSQL database is to be replaced by an easier maintainable solution.

Finally, the ultimate goal is to revive the project and bring those apps to the App Store and Google Play Store.

1.3. State Of The Art

In order to tackle the problem of having missing or outdated data in OSM, several tools exist. However, these tools are not easy to work with or require some previous knowledge about OSM tags. Nevertheless, the idea of completing missing OSM data through crowdsourcing or gamification has been done in several projects.

One example is *MapRoulette*[1]. It is a website which presents the user challenges which he or she can solve. For instance, one challenge is *Equivalent Ways*. The user has to determine if two or more ways connect the same nodes with the same shape and correct it. The user can choose to correct the error, ignore it or mark it as false positive. Once the user has solved the error, the next challenge begins as the roulette wheel turns again.

[1] http://www.maproulette.org/

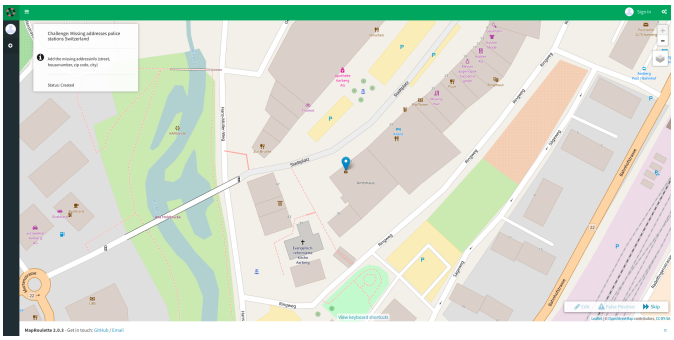

Figure 1.1.: Map Roulette

Another example which makes more use of gamification is the Android app *StreetComplete*[2]. The found issues/quests are presented to the user as markers on a map (like i.e. in Osmose) so that each is solvable by filling out a simple form to complete/correct the information on site. There is no backend, since the user logs in with his or her OSM account. The errors are directly fetched by the Overpass API. Also, solved quests are directly written back to OSM with the associated user account. This might be confusing, since it is rather an editor than a game[3]. This project was discovered right after the development of Kort Native, which is important to state, because the projects are somewhat similar.

Currently the following mission types are available in StreetComplete:

Mission Description	**OSM Tag**
diaper changing table	diaper
bicycle parking capacities	amenity=bicycle_parking, capacity
bicycle parking cover	amenity=bicycle_parking, covered
missing levels	building:levels
bus stop shelter	highway=bus_stop, shelter
housenumbers	addr:housenumber
missing speed limit	maxspeed
place without opening hours	opening_hours
place names	name
street without a name	name
road surfaces	surface
roof shapes	roof:shape
sport pitch	leisure=pitch, sport

[2] https://github.com/westnordost/StreetComplete
[3] https://lists.openstreetmap.org/pipermail/talk/2017-June/078108.html

tactile pavings on bus stops	(highway=bus_stop, tactile_paving
toilets fee	amenity=toilets, fee
wheelchair access	wheelchair

Table 1.1.: Street Complete Missions

For a complete list of Kort mission see chapter 9.3.

One last example which was also found at the end of this project is called *Google Local Guides*. It is a new feature in Google Maps for Android[4]. It can be accessed under *Your contributions/Improve the map near you*. There are two types of missions:

- **Uncover missing info:** Find places nearby that need information, like hours, website, or photos.
- **Check the facts:** Verify information that other people suggested about places nearby.

Google uses this data to improve Google Maps by presenting missing information of places as missions.

Each time a contribution is made, one will earn points. Those points help reaching higher levels in the Local Guides Program. At a certain level, one can also unlock Local Guide badges. As a reward one gets access to special perks and early access to Google features. With a separate forum called Local Guides Connect, one can meet other people who share the same pastime[5].

[4] https://maps.google.com/localguides/
[5] https://www.localguidesconnect.com/

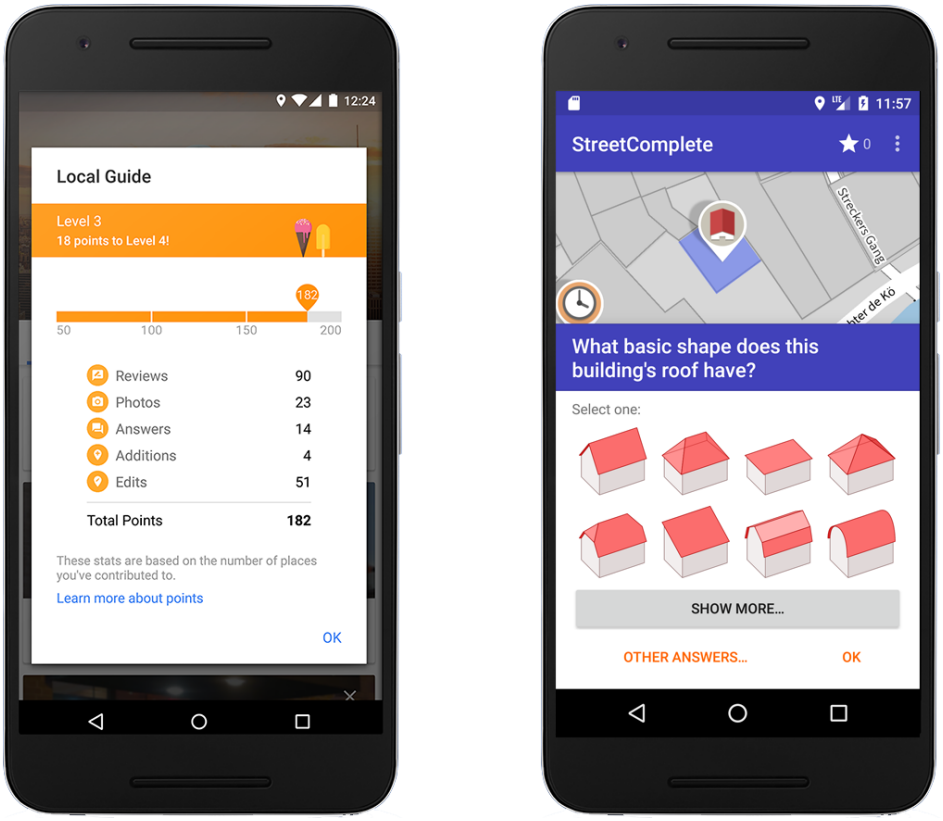

Figure 1.2.: Local Guides & StreetComplete

2. Evaluation

This chapter covers the evaluation of technologies to satisfy the requirements. Furthermore, some frameworks and architectures are highlighted.

2.1. Technologies

There are three types of mobile apps that developers can write: Native Apps, Web apps, and Hybrid Apps. The latter make heavy use of embedded webviews with HTML and JavaScript. The contenders in this area are *Apache Cordova*[1] (FKA PhoneGap), *Ionic*[2] and *Appcelerator Titanium*[3]. These facilitate the development using web technologies and frameworks while having properties of native apps (e.g. offline functionality, hardware access). The whole app is packaged as a native app which allows them to be available on the App Stores.

Since the requirements of this work state building a native but cross-plattform app, these frameworks are ineligible. However, there exist also native cross platform solutions, some of which shall be presented at this point.

2.1.1. Xamarin

Xamarin is a Microsoft-owned company founded in 2011. Their IDE Xamarin Studio allows true native app development in C# but add-ins to Microsoft Visual Studio are also supplied. Xamarin supports developing for Android, iOS as well as Windows Phone. It respects the uniqueness of each platform and does not compel developers to design apps that look the same on all platforms. On the contrary, it encourages them to take advantage of the unique features of each platform [24]. Betting on a popular language like C#, Xamarin allows to share most of the code across platforms. This includes for instance code for web services, databases or business logic in general. However, the UI for each platform needs to be created individually. All the APIs available in Android and iOS are available through regular C# class libraries. In the end, Xamarin apps are compiled to native binaries, so that they are able to perform without any performance degradation.

[1] https://cordova.apache.org/
[2] https://ionicframework.com/
[3] https://www.appcelerator.com/

2.1.2. Progressive Web Apps

Progressive Web Apps or PWA are like mobile websites, however, they offer a richer experience comparable to a Native App. Once the page is loaded, service workers enable the use in offline mode. Also, loading times are faster than on a regular mobile website. Finally, PWA can be added to the homescreen and offer an immersive full screen experience with help from a web app manifest file. As a big advantage, PWA support web push notifications[4].

This type of app seems to be the jack of all trades. Not quite. Unfortunately, most of these features are not supported on iOS yet. Only time will tell if Apple will allow such kind of apps on iOS.

2.1.3. React Native

React Native is based on the well known *React* framework by *Facebook* and *Instagram*. It is a JavaScript library for building user interfaces and has been introduced in 2015. Since then, it received regular updates (sometimes every two weeks) as well as great support from the community. It shares many properties with its parent framework, some of which shall be highlighted at this point.

React forces developers to break their application down into discrete components, each representing a single view. Like with LEGO bricks, components are extremely easy to use and reusable. They can be nested and combined and become declarative and predictable UIs.

Instead of using a regular DOM, *React* uses a Virtual DOM. So instead of traversing the whole DOM, *React* only re-renders the parts of the Virtual DOM that changed. This can be efficiently done by using a unique id for each component. Whenever there is a *setState* call on a component, React will mark it as dirty as depicted in figure 2.1. This process is known as batching and means that during an event loop, there is exactly one time when the DOM is being updated, which makes it capable of high-performance processing. Afterwards the dirty components rebuild their virtual DOM for their children. This process is known as sub-tree rendering [12]. It is even possible to exclude children from this process programmatically (Selective-sub-tree Rendering)(figure 2.2).

[4] https://developers.google.com/web/progressive-web-apps/

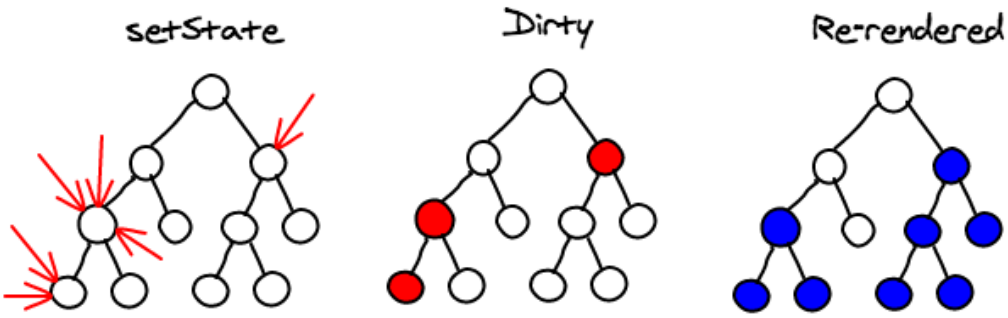

Figure 2.1.: React Virtual DOM: Batching & Sub-Tree Rendering [12]

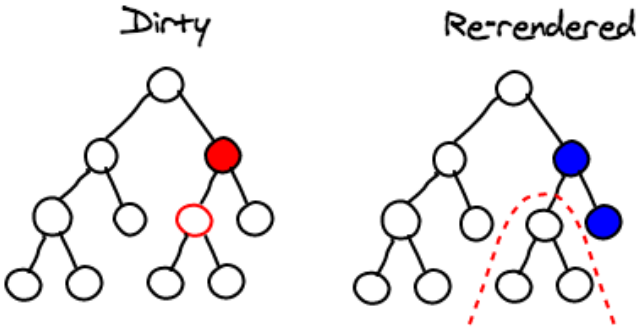

Figure 2.2.: React Virtual DOM: Selective Sub-tree Rendering [12]

Another notable feature known to *React* is *One-way data flow*. A React component cannot directly modify any of its properties. Instead, callback functions are passed which modfiy values. This mechanism is also described as 'properties flow down, actions flow up'. There is a separate subchapter covering state management in React (see 2.2.1).

Finally, *React* uses JSX, an extension to JavaScript. It is basically syntactic sugar for React functions, but is really handy to use, since it simplifies the code dramatically. The use is similar to HTML tags [15]. The snippets in the following listing 2.1 are identical in their function.

```
<MyButton color="blue" shadowSize={2}>
  Click Me
</MyButton>

// compiles into:

React.createElement(
  MyButton,
  {color: 'blue', shadowSize: 2},
  'Click Me'
)
```

Code Snippet 2.1: React Native Component

React Native has all those properties. The beauty of *React* is that it is not tightly coupled to the DOM. *React* can wrap any imperative view system, like UIKit on iOS. So, instead of running the code in a browser and rendering *divs* and *spans*, it is run in an embedded instance of JavaScriptCore inside the mobile app while rendering happens in higher-level platform-specific components [28]. This process has been implemented for Android and iOS and the same JavaScript code can be run natively on both platforms. It looks like support for the Windows platform is underway.[5]

In terms of performance, *React Native* easily outplays its Hybrid App counterparts. In release mode, it achieves a solid 60 frames per second resulting in buttery smooth animations. The main part of a *React Native* app is the JavaScript thread. This is where API calls are made, touch events processed and business logic run. At the end of each iteration of the event loop, updates to the native views are batched and sent over to the native side. This way the main thread is left alone and reserved for UI.

The designing of the views is done with the help of the flexbox algorithm known from the CSS world [32]. This allows to have flexible layouts respecting the size and pixel density of each and every device.

Sometimes an app needs access to a platform API, and React Native doesn't have a corresponding module yet. Also, in case of extending an existing native app with React Native, there must be a mechanism for an interaction from native code to *React Native* and vice-versa. This mechanism exists and is called *Native Modules* [5].

One of the best features of *React Native* is the support for rapid development. Instead of recompiling, the changes are loaded instantly by hot reloading. This technology sure is compelling.

2.2. Architecture

In terms of architecture there are many possibilities to consider. For an in-depth look at the chosen architecture, see chapter 4. This sub-chapter is merely an excerpt of some of the chosen frameworks.

2.2.1. State Management in React

While *React Native* is great for writing reusable components, the question arises where to manage the state of the app. The motivation is to define how data flows through the system and makes the UI predictable. Also, there should only be one source of state and it should be updated in a controlled manner. There are many frameworks that address this issue in similar ways, some of which are presented at this point.

[5] https://blogs.windows.com/buildingapps/2016/04/13/react-native-on-the-universal-windows-platform/

Flux

When Facebook announced Flux, they just described the architecture pattern without releasing a concrete implementation. This is why Flux implementations mushroomed in recent years and it is almost impossible to stay on top of things. However, in general they are similar. The centerpiece is always an unidirectional data flow as depicted in figure 2.3.

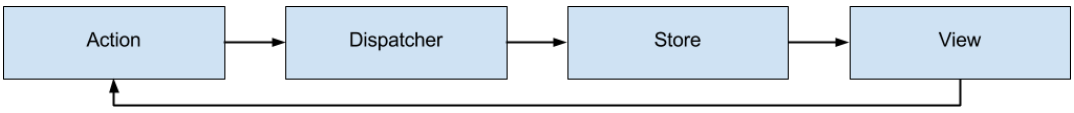

Figure 2.3.: Flux: Unidirectional Data Flow

The **Dispatcher** receives actions and dispatches them to stores that have registered with the dispatcher. Every store will receive every action.

The **Stores** hold the data of an application. They register with the application's dispatcher enabling them to receive actions.

Actions are simple objects that have a type field and some data. Basically, they represent the ways in which anything might interact with the application.

Last but not least, the data from the stores is displayed in **Views**. When a view uses data from a store it must also subscribe to change events from that store. This way, when the data changes, the view re-renders.

Some of the downsides of the concrete implementation is the clumsy implementation and the fact that it is a big fat singleton. There is an article about the criterias on which one can choose the best implmentation, but in the end only the experience tells how it performs for one's use case [13]. In scope of this work, only two implementations have been considered which shall be presented in the next subchapters.

For a direct comparison on Flux implementations there is a great repository[6]. The same app is implemented with different Flux implementations.

MobX

MobX claims to be a simple and scalable state management solution which has about half a million downloads per month (2.7.17). Conceptually, *MobX* treats the application like a spreadsheet [6].

[6] https://github.com/voronianski/flux-comparison

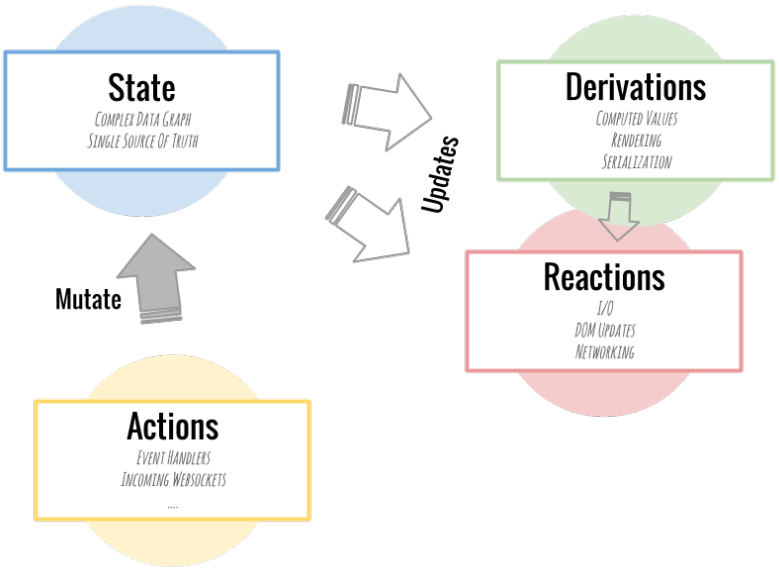

Figure 2.4.: MobX: Data Flow [6]

The 'data cells' are the application **state** (also known as single source of truth). It holds primitives, arrays and references.

All values which can be derived from the application state are called **derivations**. These can be computed values or even complex visual representation of the state. In spreadsheet terms, these are the formulas and charts.

Reactions are similar to derivations, except they do not produce any values, but rather run automatically to perform some task.

Actions are the only actors that can change the state. MobX makes sure that all changes to the state are automatically processed by all derivations and reactions.

Compared to Flux, it looks like only the terms have changed. However, there are differences especially when compared to Redux (see next section). For instance, MobX is influenced by Object-Oriented and Reactive Programming principles. As such it lets specific pieces of data define as observable. Also, it seems to have less boilerplate code.

Redux

Redux is a popular option especially in combination with *React*. It has about 3 million downloads per month (2.7.17).

Compared to Flux, Redux only has only one state object. Also, the dispatchers are now part of the Redux store. A new element added to the mix are Reducers. They expect the developer to return a copy of the state and not mutating it, which is great for debugging.

As a bonus, one gets features like undo / redo for free since the state is stored in a single tree.

Combined with React components as views, the architecture looks as follows:

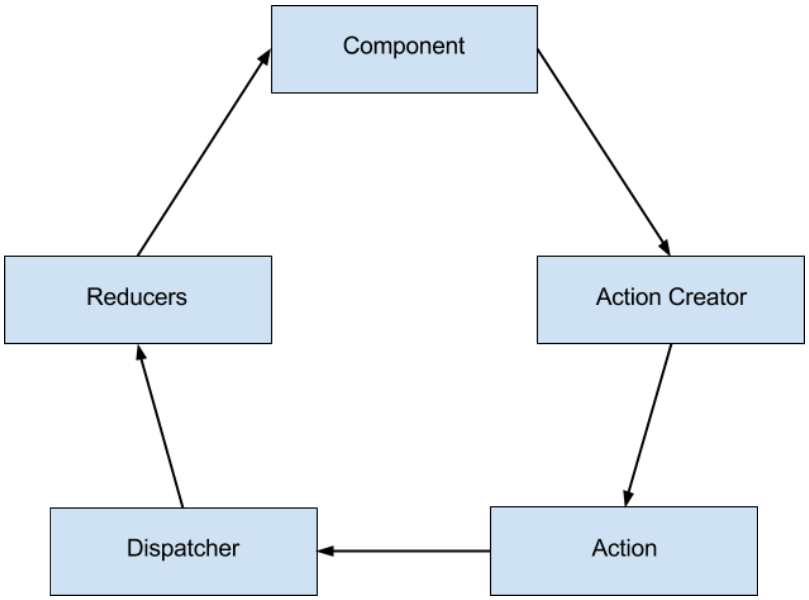

Figure 2.5.: Redux: Data Flow

Actions are payloads of information that send data from the app to the store, whereas **Action Creators** are plain functions that create such actions.

The transformation of the state tree, is done with **Reducers**. They are pure functions that take the previous state and an action, and return the next state without mutating the previous one. They can be split into several small ones, each responsible for specific parts of the state tree.

In order to have abilities like asynchronous API calls or routing, a Redux middleware is used. For instance Redux-Thunk middleware allows to write action creators that return a funtion instead of an action. The thunk can be used to delay the dispatch of an action.[7]

Depending on the requirements and experience one choses Redux over MobX or the other way around. An interesting conversation about the pros and cons can be found on Reddit.[8]

2.2.2. Docker

Docker is an open platform for developing, shipping and running applications. It enables the separation of application and infrastructure resulting in faster software delivery.

Docker uses a client-server architecture [1]. The client connects to the Docker daemon, which ultimately does the building, running and finally distributing the Docker containers. Client and daemon do not necessarily have to run on the same machine.

[7] https://github.com/gaearon/redux-thunk
[8] https://www.reddit.com/r/reactjs/comments/4npzq5/confused_redux_or_mobx/

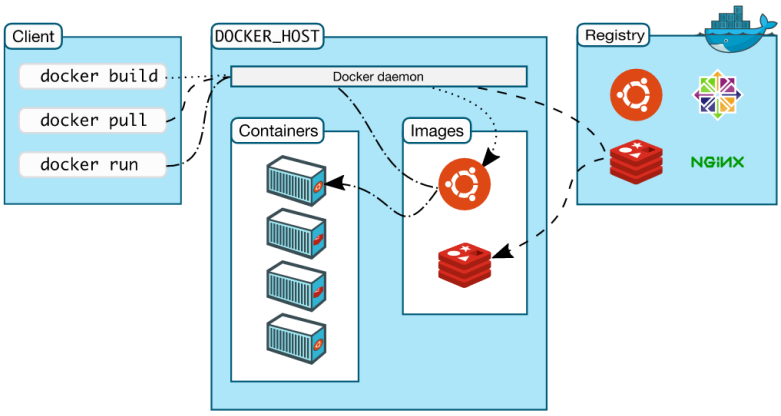

Figure 2.6.: Docker Architecture [1]

Other Docker essentials include images, registries and containers.

Docker images are read-only templates and can for instance contain already installed software such as an Ubuntu Linux and an nginx webserver. They are used for creating Docker containers. Docker images are built with the docker build command.

Docker registries are used for distributing existing Docker images. They are either private or public (the Docker Hub). Docker images are downloaded from Docker registries with the `docker pull` command.

Docker containers are built from Docker images. They contain everything needed for an application to run and have their own file system and networking. Each container is an isolated application platform and can be run, started, stopped, moved and deleted. Docker containers are run with the `docker run` command.

2.3. Other Frameworks

Some frontend and backend frameworks that have been evaluated shall be presented in this section.

2.3.1. Mapbox GL

The centerpiece of the app is the map. It has the following requirements:

- Annotations for placing missions
- Native Support (no JavaScript library)
- OSM layer

The only mapping library which satisfies these requirements is the experimental React Native

Mapbox GL library[9]. Some alternatives have been evaluated in a previous work [26], however it came to the same conclusion.

Even though the last release has been a while (17. Nov 2016), Mapbox is serious about the development of React Native Mapbox GL and the library might see its way out of the experimental status[10].

For measuring the performance of putting annotations on a map, a separate benchmark app has been written. It could be shown that the library handled 10'000 markers in reasonable time.

This framework suits the needs very well, however one should keep an eye on pricing options[11]. It states that 50'000 uniqe mobile users are free of charge. The next plan already costs $499/month, which is rather pricey for a non-commercial project.

2.3.2. Connexion

In order to have code which is easier to maintain and readable, it was decided to write the backend in *Python*. There are several frameworks which allow developing API's [2], a prominent one being *Flask*. It is a microframework, meaning it keeps the core simple and extensible.

The wish to have a nice and clean up-to-date documentation of the interface, lead to a framework which is fairly new: *Connexion* by *Zalando*. It also uses *Flask* in the background but what it sets apart from other frameworks is the way it handles API specification. It allows to write the Swagger specification first, then maps the endpoints to Python functions. Most other frameworks generate specification on existing code. This approach is rather nice and helps to simplify the development process while guaranteeing that the API meets specification.

[9] https://github.com/mapbox/react-native-mapbox-gl
[10] https://twitter.com/incanus77/status/868871838478237696
[11] https://www.mapbox.com/pricing/

Some of the selling points according to the documentation[12] are the following:

- Validates requests and endpoint parameters automatically, based on your specification
- Provides a Web Swagger Console UI so that the users of your API can have live documentation and even call your API's endpoints through it
- Handles OAuth 2 token-based authentication
- Supports API versioning
- Supports automatic serialization of payloads. If your specification defines that an endpoint returns JSON, Connexion will automatically serialize the return value for you and set the right content type in the HTTP header.

OAuth support and Web Swagger Console UI are a welcoming addition to the requirements. Finally, automatic serialization helps to ensure correct content types which will be JSON.

[12] https://github.com/zalando/connexion

3. Requirements Engineering

This chapters covers the analysis of the use cases and presents the sequence and activity diagrams. It concludes with the non-functional requirements.

3.1. Use Cases

The following figure 3.1 presents the use cases and how they interoperate with each other.

Each Use Case is then described in detail.

Use Case	**UC1**
Use Case Name	fetch missions
Use Case Description	The player wants to fetch new mission either around him or her or at a desired location.
Actor	Kort Player
Pre-condition	Missions are available for that location on Kort server.
Basic Flow	The following scenarios trigger this action: • The app receives a new GPS fix • The player wants to fetch new missions around the current map center
Post-condition	The mission annotations appear on the map.
Alternative Flows	An error occurred and an appropriate message is displayed.

Table 3.1.: UC1: fetch missions

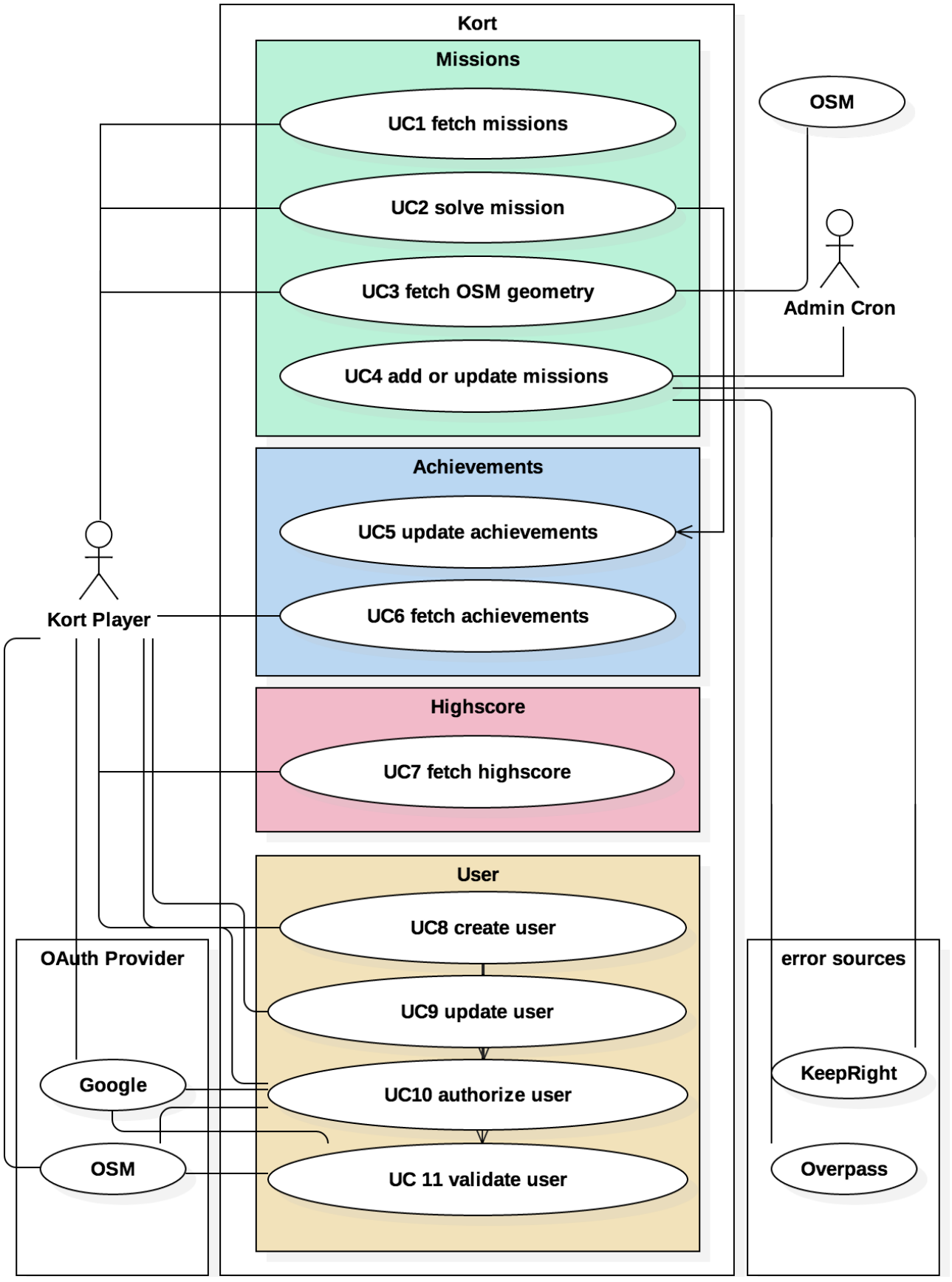

Figure 3.1.: Use Cases

Use Case	**UC2**
Use Case Name	solve mission
Use Case Description	The player submits a solution to a given mission.
Actor	Kort Player
Pre-condition	The provided answer was validated on the client.
Basic Flow	The following steps are part of this use case: • The mission id of the solution is validated. • The answer is written to the database. • Check whether new achievements have been obtained. • Return new achievements.
Post-condition	A new mission fetch is initiated.
Alternative Flows	An error occurred and an appropriate message is displayed.

Table 3.2.: UC2: fetch missions

Use Case	**UC3**
Use Case Name	fetch OSM geometry
Use Case Description	The geometry of a specific OSM feature is retrieved.
Actor	Kort Player
Pre-condition	The player tapped on a mission annotation.
Basic Flow	A request to the OSM API is made. The returned coordinates are transformed to the format which Mapbox can handle.
Post-condition	The geometry could be visualized on the map.
Alternative Flows	An error occurred and an appropriate message is displayed.

Table 3.3.: UC3: fetch OSM geometry

Use Case	**UC4**
Use Case Name	add or update missions
Use Case Description	New missions are added to the database.
Actor	Admin / Cron
Pre-condition	The error sources (KeepRight, Overpass) are online and reachable.
Basic Flow	The data is loaded from the error sources and the new data is being added to the error database.
Post-condition	More missions are available to the Kort player.
Alternative Flows	An error occurs. It will be logged.

Table 3.4.: UC4: add or update missions

Use Case	**UC5**
Use Case Name	update achievements
Use Case Description	Check if new achievements have been obtained.
Actor	Kort Player
Pre-condition	A valid solution has been submited.
Basic Flow	It is checked whether new achievements have been obtained. If so, the new achievements are persisted and finally returned to the player for display.
Post-condition	The player receives the newly obtained achievements.
Alternative Flows	An error occurred and an appropriate message is displayed.

Table 3.5.: UC5: update achievements

Use Case	UC6
Use Case Name	fetch achievements
Use Case Description	The player wants to fetch all achievements he or she obtained.
Actor	Kort Player
Pre-condition	The player is logged in. If not, no achievements are listed as obtained.
Basic Flow	On request, all possible achievements are fetched from the database and presented to the user. The ones that are already obtained are flagged appropriately.
Post-condition	All the achievements appear on the players app.
Alternative Flows	An error occurred and an appropriate message is displayed.

Table 3.6.: UC6: fetch achievements

Use Case	UC7
Use Case Name	fetch highscore
Use Case Description	The player wants to fetch the current leaderboards.
Actor	Kort Player
Pre-condition	The player is logged in. If not, one will not appear on the leaderboard.
Basic Flow	On request, the appropriate leaderboard is loaded and presented.
Post-condition	All the leaderboards appear on the players app.
Alternative Flows	An error occurred and an appropriate message is displayed.

Table 3.7.: UC7: fetch highscore

Use Case	**UC8**
Use Case Name	create user
Use Case Description	The player wants to create a new account by logging in with Google or OSM OAuth.
Actor	Kort Player
Pre-condition	The user is successfully authorized (UC10) and validated (UC11)
Basic Flow	When both UC10 and UC11 return success, a new user is created in the database. A generated and unique secret token is then returned to the player which will be the authorization key from now on.
Post-condition	The player received a secret token.
Alternative Flows	An error occurred and an appropriate message is displayed.

Table 3.8.: UC8: create user

Use Case	**UC9**
Use Case Name	update user
Use Case Description	The player wants to update his or her user profile.
Actor	Kort Player
Pre-condition	A user account for this player exists in the database.
Basic Flow	This use case is triggered by two events: • The player wants to update his or her user profile, in order to get his or her latest statistics or user information. • The player once logged out and wants to log in again.
Post-condition	The player has got the latest statistics and user information on the app.
Alternative Flows	An error occurred and an appropriate message is displayed.

Table 3.9.: UC9: update user

Use Case	UC10
Use Case Name	authorize user
Use Case Description	The player wants to log in with Google or OSM OAuth.
Actor	Kort Player
Pre-condition	The player has either a Google or an OSM account.
Basic Flow	The player wants to login in with the Kort app. In order to do that, he or she gets redirected to Google or OSM OAuth provider which returns a OAuth token on success.
Post-condition	An OAuth token has been issued by the OAuth provider.
Alternative Flows	An error occurred and an appropriate message is displayed.

Table 3.10.: UC10: authorize user

Use Case	UC11
Use Case Name	validate user
Use Case Description	The player wants to log in with Google or OSM OAuth. For that he or she must be validated.
Actor	Kort Player
Pre-condition	The user is successfully authorized (UC10).
Basic Flow	With the OAuth token retrieved in UC10, the user is validated and on success, user profile information is written to the database.
Post-condition	The user is successfully validated.
Alternative Flows	An error occurred and an appropriate message is displayed.

Table 3.11.: UC11: validate user

3.2. Sequence Diagrams

The following sequence diagram emphasizes the flow of control and data in the system, specifically during the user creation.

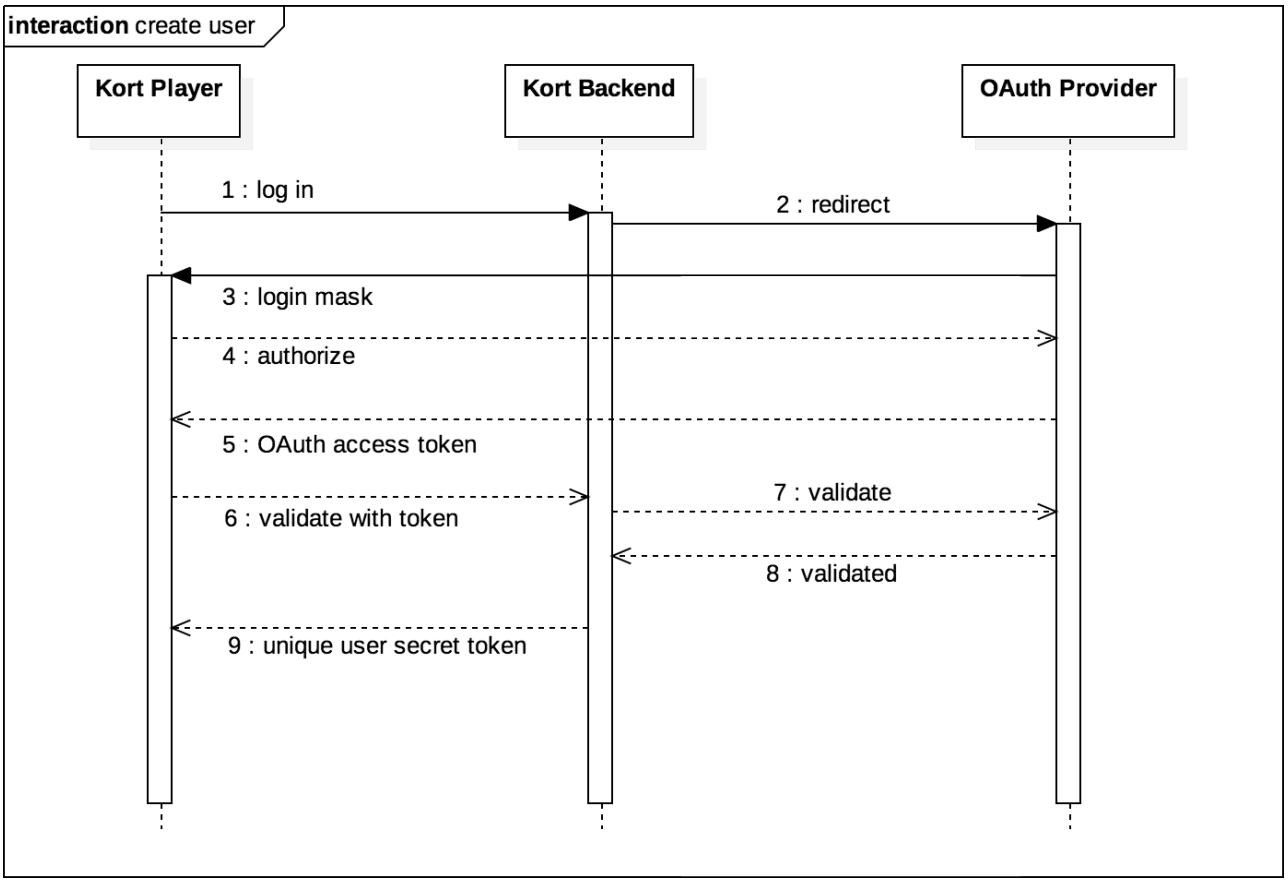

Figure 3.2.: sequence: create user

3.3. Activity Diagrams

The following activity diagram shows the actions involved with 'solve mission'.

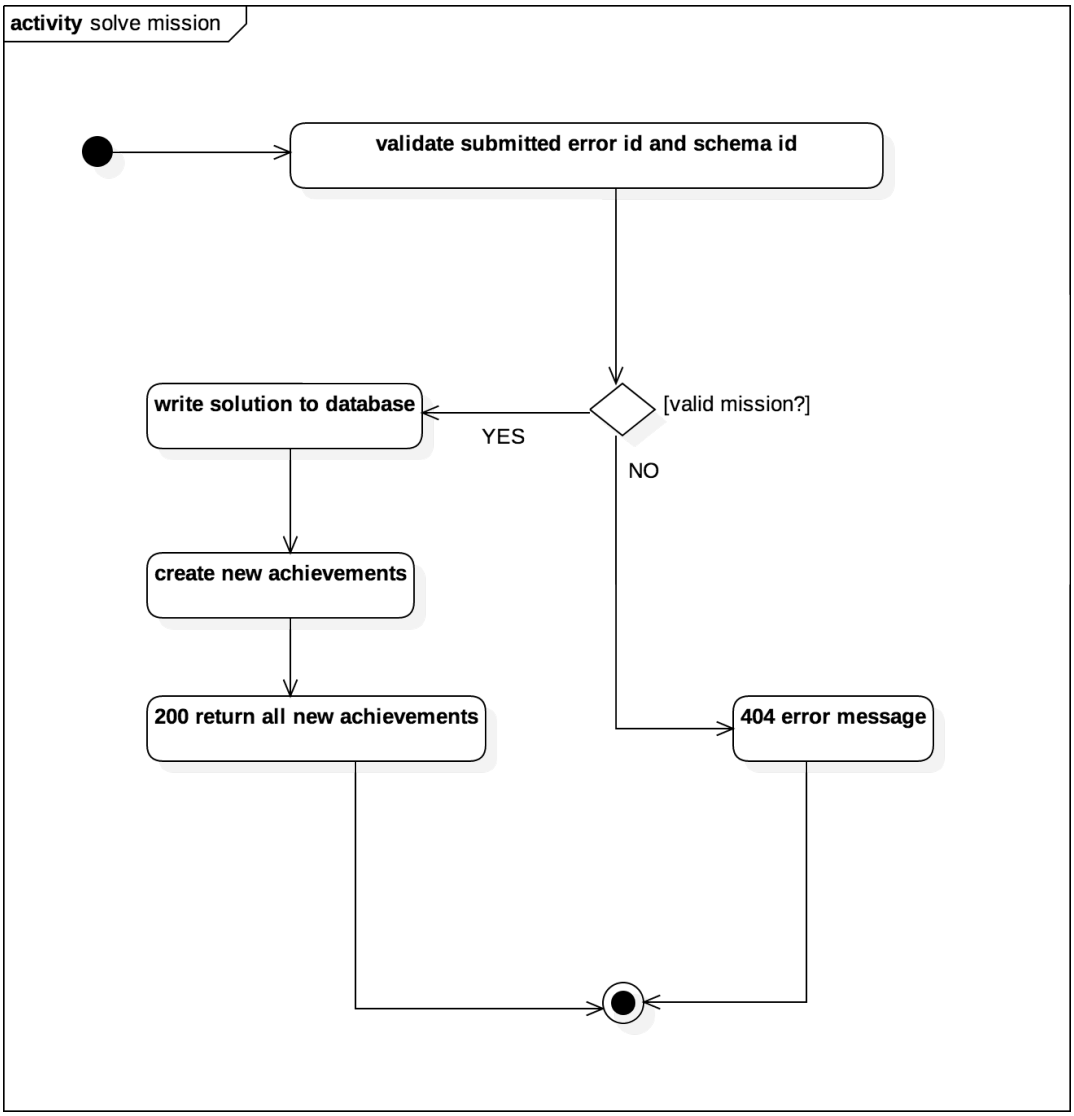

Figure 3.3.: activity: solve mission

The following activity diagram shows the actions involved with 'fetch OSM geometry'.

Since one requirement is to visualize the current osm feature on the map and the Mapbox SDK needs geometries as GeoJSON, a separate service fetches this information and constructs a response object accordingly. Note that the Mapbox SDK does not support multi polygons. Therefore, such geometries are constructed as polygons, only considering the 'outer' part.

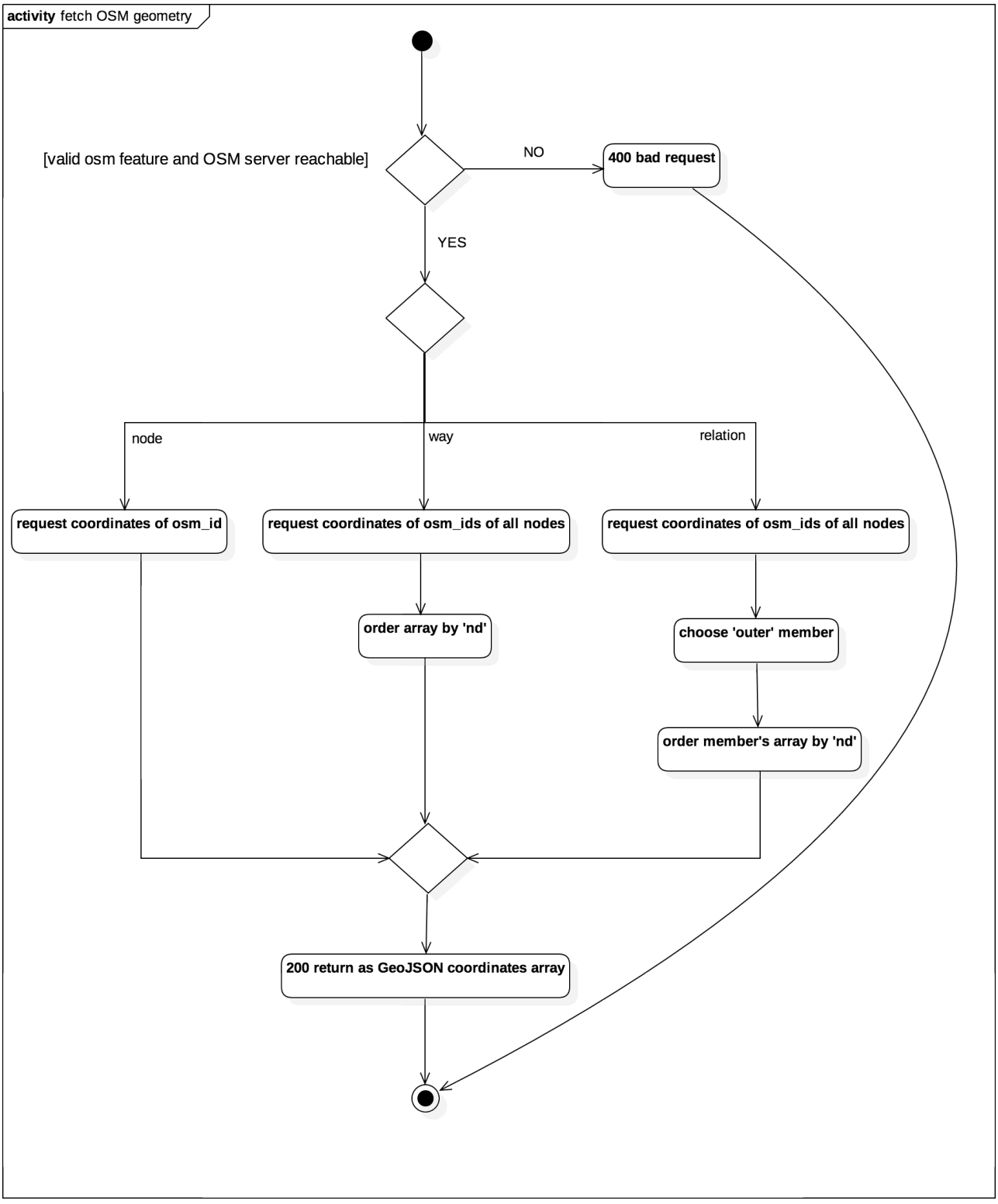

Figure 3.4.: activity: fetch OSM geometry

3.4. Non-Functional Requirements

The following quality attributes should be taken into account:

ID	Description
NFR1	The missions presented to the user must be easily solvable. For instance, drawing geometries is considered to be hard, whereas selecting predefined answers is easy.
NFR2	Fetching new missions should not take more than 3 seconds.
NFR3	There should be a variety of different mission types available.
NFR4	Internationalization (I18n): The app should be available at least in English and German with the possibility of easily adding new languages.
NFR5	No sensitive user data should be persisted and all traffic should be handled over HTTPS.

Table 3.12.: Non-Functional Requirements

4. Architecture

This chapter explains how and which architectural decisions were made. There is a distinction between frontend and backend since different technologies were applied. Finally, third-party systems are listed along with a risk assessment.

4.1. System Landscape

The backend consists of several Docker containers, which allow easy setup on different systems. It is also known as *Kort Core*. The following figure visualizes the system landscape. It also includes *Kort Native* which represents the frontend.

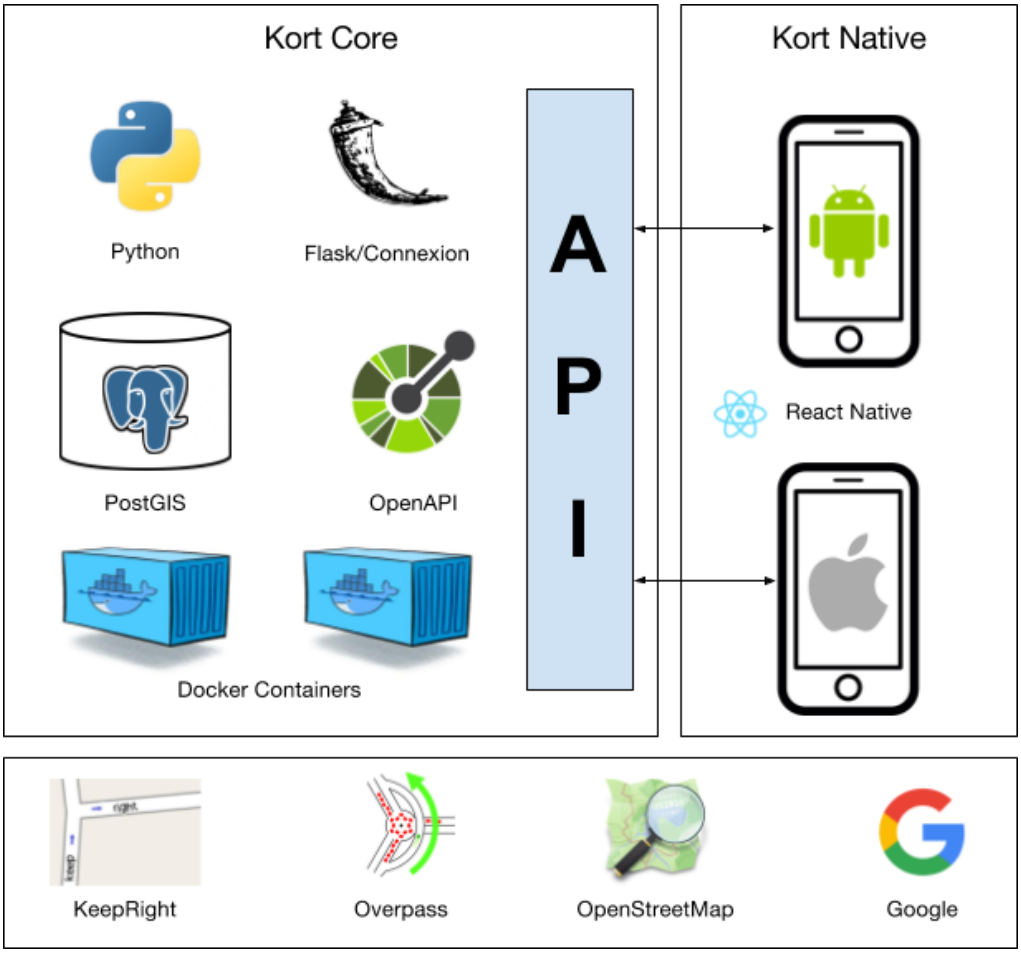

Figure 4.1.: Kort System Landscape

4.2. Y-Approach

In deciding how much design rationale is enough, the Y-approach has been followed [31]. It helps capturing a decision by providing a skeleton of statements.

One example of this approach is the following statement:

Architectural Decision #1

In the context of 'UC1: fetch missions', facing 'NFR2', we decided for PostGIS spatial queries to achieve a quick response.

This approach is compatible with the KISS principle, which was also applied in scope of this work.

4.3. Frontend

The frontend is the React Native app. This section covers the specific app components, how the state management is handled, app dependencies, and last but not least, platform specific adjustments.

Since *React Native* enables developers to write platform specific code by using the *android.js* or *ios.js* extension, the advantage of having one single codebase is lost. Therefore, it was decided to omit this feature where possible. In fact, it is only used as an entry point for each platform.

Architectural Decision #2

In the context of having a single codebase, facing code maintainability, we decided for using React Native and no platform specific code where possible to achieve creating native apps.

4.3.1. App Components

As already stated in 2.1.3, React Native apps consist of components. According to Dan Abramov, co-author of Redux, there are two types of components [8]:

- Presentational Components
- Container Components

In general *presentational components (PC)* are concerned with how things look whereas *container components (CC)* handle how things work. PC have no dependencies on the rest of the app, while CC provide the data and behavior to PC and call flux actions and provide these as callbacks to PC. Quite often PC tend to be stateless and CC tend to be stateful.

However, while this approach is optional and up to the developer, it enforces better reusability.

The following table gives an overview over the used components in the Kort app and reflects the subfolders in folder *components*:

Component Folder	**Content & Description**
common	These are all the general, presentational components. Examples include different flavors of buttons (e.g. with text, icon, image), custom views (e.g. webview, keyboard avoiding view), views for text input, popup views, and modal views among others.
achievements	All components here are related to achievements. The *AchievementsOverview* is the main container component featuring a list view consisting of presentational *AchievementsItems*.
highscore	Here are all components related to highscore. The *HighscoreOverview* is the main container component featuring multiple list views *HighscoreList*. The latter consists of *HighscoreListItems*.
login	All components related to login are here. The *LoginOverview* is the main container component. It consists of another container component *LoginBox*. The latter finally consists of the login buttons and their action calls.
mission	In this folder there are all components related to missions. Since this is the main part of the app, there are more components, some of which are described as follows: • *MissionOverivew* is the main container component. It keeps together all the components used for missions. • *Map* naturally consists of the Mapbox component as well as the buttons on the map. All map interactions such as zooming, rotating, jumping, annotation clicking, as well as switching to fullscreen are handled here. • *MissionView* is a presentational component consisting of a custom and animated sliding view. • *MissionSolver* is a container component responsible for rendering the mission details as well as handling all the actions for the current mission. • In subfolder *types* there is a component *OpeningHours* providing all the functionality and views for entering opening hours in a simple fashion.

profile	*ProfileOverview* is the main container component showing user information as well as providing the actions for login, logout and update.
settings	*SettingsOverview* is the main container component, consisting of a settings list that provides switches for setting on and off certain settings. It also navigates to a presentational *About* page as well as the showcase.
showcase	*Showcase* is a presentational component consisting of *ShowcaseItems* which are arranged in pages.

Table 4.1.: App Components

4.3.2. State Management

It was decided to use Redux as a state management library, because of its good reputation in the React community. Also, it is said to have good maintainability over a long period of time, while being well suited for large projects [27].

Architectural Decision #3

In the context of state management in React, facing maintainability, readability, testability and understandability, we decided for using Redux to achieve managing state of React user interfaces.

As already pointed out in chapter 2.2.1, Redux uses **Action Creators** and **Reducers** for handling state. In the Kort app, these parts are reflected in the folders *actions* and *reducers*. All actions can be found in the file *types.js* in the folder *actions*. The action creators that share the same reducer are then put in a separate file. Chapter 6.1.2 covers these in more depth.

4.3.3. App Dependencies

A number of software libraries have been used. A good source for libraries is the NPM website[1], as well as the Awesome React Native repository on GitHub[2].

One pivotal library in this work of course is the React Native Mapbox GL. Based on the evaluation in chapter 2.3.1 as well as evaluations conducted in one other paper [26], it was decided to use this library.

[1] https://www.npmjs.com/
[2] https://github.com/jondot/awesome-react-native

Architectural Decision #4

In the context of visualizing an interactive map, having current OSM map layers, we decided for using React Native Mapbox GL to achieve putting annotations on an interactive map, accepting the downside of having an experimental and currently not supported library.

Other used libraries are the following:

Library	Description
lodash[3]	The Swiss Army knive for JavaScript. A useful utility library delivering modularity, performance & extras.
opening_hours[4]	Library to parse and process the opening_hours tag from OpenStreetMap data. It is used for validation of free text input.
react-native-animatable[5]	Easy to use declarative transitions and a standard set of animations for React Native.
react-native-google-signin[6]	Google Signin implementation for React Native.
react-native-hyperlink[7]	A `<Hyperlink />` component for react-native that makes urls, fuzzy links, emails etc. clickable.
react-native-i18n[8]	Integrates I18n.js with React Native. Uses the user preferred locale as default.
react-native-modal-datetime-picker[9]	This library exposes a cross-platform interface for showing the native date and time pickers inside a modal.
react-native-modal-dropdown[10]	A react-native dropdown/picker/selector component for both Android & iOS.
react-native-page-control[11]	Page Control for React Native, like iOS UIPageControl, APIs are same as UIPageControl.
react-native-rest-client[12]	Simplify the RESTful calls of your React Native app.

[3] https://lodash.com/
[4] https://github.com/opening-hours/opening_hours.js
[5] https://github.com/oblador/react-native-animatable
[6] https://github.com/devfd/react-native-google-signin
[7] https://github.com/obipawan/react-native-hyperlink
[8] https://github.com/AlexanderZaytsev/react-native-i18n
[9] https://github.com/mmazzarolo/react-native-modal-datetime-picker
[10] https://github.com/sohobloo/react-native-modal-dropdown
[11] https://github.com/silentcloud/react-native-page-control
[12] https://github.com/javorosas/react-native-rest-client

react-native-router-flux[13]	This library handles navigation between views. It also features navigation and tab bars.
react-native-segmented-control-tab[14]	A react native component with the same concept of react native's SegmantedControlIOS, Primarily built to support both IOS and Android.
react-native-select-multiple[15]	A customiseable ListView that allows you to select multiple rows.
react-native-settings-list[16]	A clean and highly customizable React Native implementation of a list of settings for a settings page.
react-native-simple-store[17]	A minimalistic wrapper around React Native's AsyncStorage.
react-native-vector-icons[18]	Customizable Icons for React Native with support for NavBar/TabBar/ToolbarAndroid, image source and full styling.
react-native-version-number[19]	Returns the CFBundleShortVersionString and the CFBundleVersion. For Android, returns the versionName and versionCode.
redux-thunk[20]	Redux Thunk middleware allows you to write action creators that return a function instead of an action. The thunk can be used to delay the dispatch of an action, or to dispatch only if a certain condition is met.

Table 4.2.: JavaScript Libraries

The current version used for these libraries can be looked up in the *package.json* file.

4.3.4. Platform Specific Adjustments

In order to get certain things to work, platform specific adjustments needed to be applied.

One example is providing an interface for other apps to open the app with certain parameters and payloads. This mechanism is called Inter-App Communication. The most prominent way to do this is Deep Linking. One defines URL schemes which are supported by the app and maps these to certain actions within the app. In the case of the Kort app this is especially useful to create a smooth OAuth flow.

[13] https://github.com/aksonov/react-native-router-flux
[14] https://github.com/kirankalyan5/react-native-segmented-control-tab
[15] https://github.com/tableflip/react-native-select-multiple
[16] https://github.com/evetstech/react-native-settings-list
[17] https://github.com/jasonmerino/react-native-simple-store
[18] https://github.com/oblador/react-native-vector-icons
[19] https://github.com/APSL/react-native-version-number
[20] https://github.com/gaearon/redux-thunk

Architectural Decision #5

In the context of login via OAuth, facing different OAuth providers, we decided for using Deep Linking to achieve a smooth login experience.

The following URL schemes are currently in place:

URL	Parameters	Usage, Action
`com.googleusercontent.apps.<uniqueID>:/oauth2callback?code=<OAuthCode>`	**uniqueID:** the client ID registered for this OAuth account **OAuthCode:** the authorization code associated with this OAuth account	Once the login with Google is successful, it will open the Kort app with the specified URL. The Google API then provides a user object with a unique user token id. This token is then validated with the Kort backend as seen in figure 3.2.
`kortapp://payload?secret=<userSecret>&userId=<userId>`	**userSecret:** the unique user secret associated with the user id **userId:** the unique user id in the kort backend	Once the login with OSM is successful, it will open the Kort app with the specified URL. The URL is then parsed according to the given scheme. With the user id and the user secret which are persisted on the device, it allows faster login from now on.

Table 4.3.: URL Schemes

One could also think of using Deep Linking for pointing the map inside Kort directly to given coordinates. However, this was not implemented.

The specific adjustments applied on each platform can be looked up in chapter 6.1.3.

4.4. Backend

This section covers the Docker components and the underlying data model. It concludes with how the API is designed.

4.4.1. Docker Components

Kort Core is built with *Docker* and is therefore shipped in containers. This allows to have an extra layer of abstraction and avoids overhead of a real virtual machine. Specifically, it is built with *Docker Compose* thus allowing to define a multicontainer architecture defined in

a single file.

Architectural Decision #6

In the context of backend deployment, facing maintainability, testability and portability, we decided for using Docker Compose to achieve low complexity and easy deployment.

The Docker Compose YAML file looks as follows:

```
api:
  restart: always
  build: ./src/api
  expose:
    - "8000"
  links:
    - postgres:postgres
    - tokeninfo:tokeninfo
  volumes:
    - /usr/src/app/static
  env_file:
    - .env
    - .envTest
  command: /usr/local/bin/gunicorn -w 2 -b :8000 app:app

tokeninfo:
  restart: always
  build: ./src/api
  expose:
    - "7979"
  links:
    - postgres:postgres
  volumes:
    - /usr/src/app/static
  env_file: .env
  command: /usr/local/bin/python3 tokeninfo.py

nginx:
  restart: always
  build: ./src/nginx/
  ports:
    - "80:80"
  volumes:
    - /www/static
  volumes_from:
    - api
  links:
    - api:web

data:
  image: mdillon/postgis
  volumes:
    - /var/lib/postgresql
```

```
44    command: "true"
45
46  postgres:
47    restart: always
48    build: ./database/
49    env_file: .env
50    volumes_from:
51      - data
52    ports:
53      - "5432:5432"
```

Code Snippet 4.1: Docker Compose

First, the data-only container pattern is applied for the **data** container which stores all the database files. This container is then mounted from the **postgres** which is responsible for data querying as well as importing data. Second, the **api** container provides a REST interface to the outside world. In order to look up user tokens, there exists a separate container **tokeninfo**. Finally, **nginx** represents a proxy server which maps port 80 to the APIs port. This container is only needed for development and not in a production environment which often times provides its own proxy.

Note the *.env* and *.envTest* files. More on this in chapter 6.2.3. The single containers can be started separately but the whole environment can be easily brought up with a single docker `compose up` command. Dockerfiles meet best practices defined by Docker [14].

4.4.2. Data Model

From an early point on, it was decided to extend the old data model for various reasons. First of all, since data is a valuable good, it is vital to transfer old data from the old implementation. Second of all, some of the import scripts could be reused, thus reducing some development effort.

Architectural Decision #7

In the context of data modelling, facing adaptability, we decided for extending the old Kort database design to achieve the ability for easier migration of old data and the opportunity to reuse old import scripts.

The data model is visualized in figure 4.2. Some fields have been replaced or moved. For instance, in the *user* table it makes no sense to have a field for Koin count, but rather calculate this value on request.

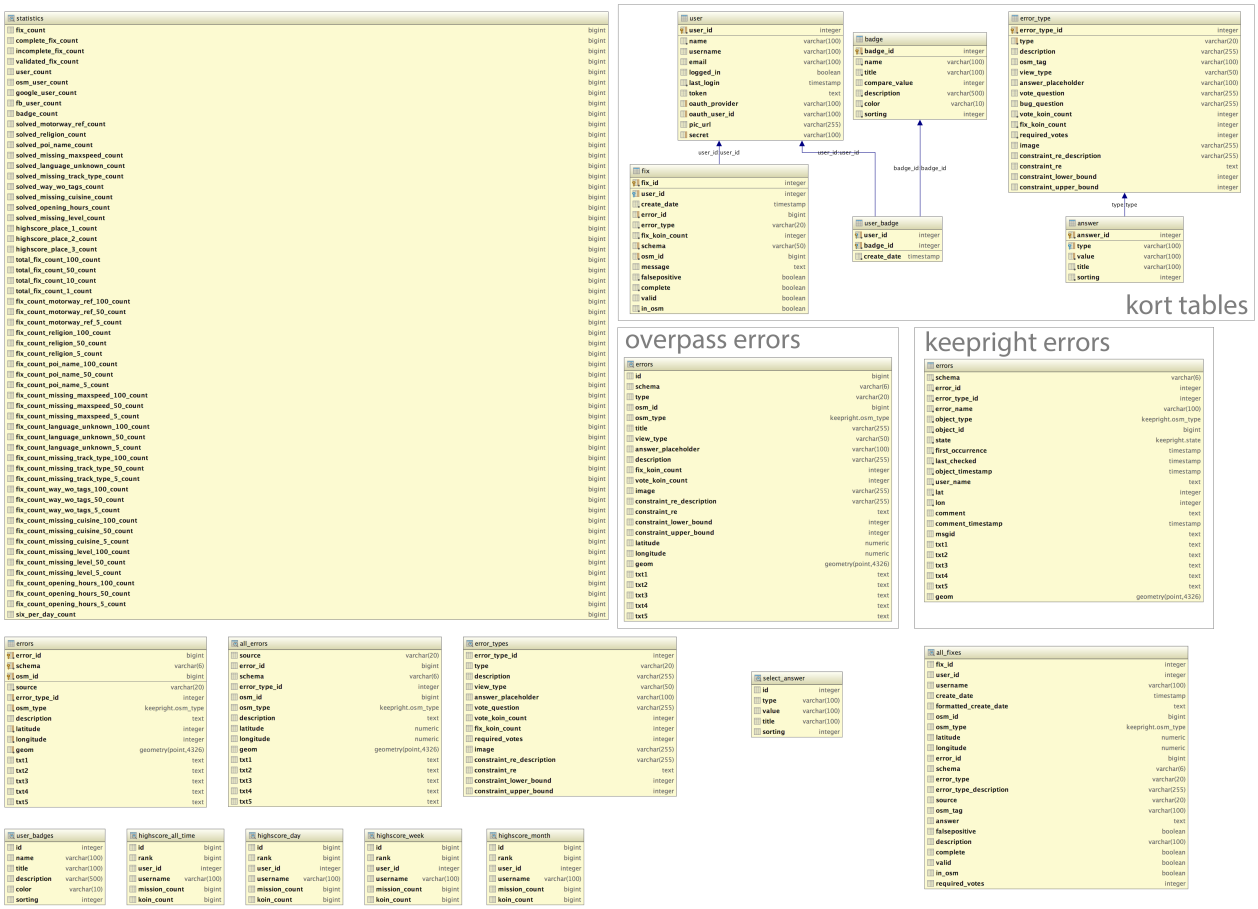

Figure 4.2.: Kort data model

Figure 4.2 depicts all views and tables. In order to decouple dependencies, there is a separate schema for KeepRight and Overpass. The Kort tables are highlighted appropriately. The rest are views.

4.4.3. API

Since time was of the essence in this project, the need for simple and small backend services was evident. Using microservices with a well crafted API simplifies the development process dramatically. Furthermore, failing to describe an API in a clear manner as well as maintaining its documentation leads to a broken system. That is why a so-called *API First* approach has been followed.

Architectural Decision #8

In the context of API design, facing readability and clarity, we decided for an API First approach to achieve an up-to-date documentation and specifications as a first-class artifact.

Writing the specification of a service upfront leads to discussions among possible consumers of the API. Furthermore, it enables the creation of mocks of the API and documentation before any line of code has been written [35]. Similar to the *mobile first* approach, where everything revolves around the idea of the product being consumed by mobile devices, the *API first* approach focuses on the API consumed by client applications [19].

In terms of webservices, it was decided to use RESTful services, since they are really simple to use.

Architectural Decision #9

In the context of client communication, facing robustness, simplicity, usability and scalability, we decided for using REST services to achieve performant client-server communication.

In order to describe the API as intended with the *API first* approach, multiple options exist: There is the known *API Blueprint*[21] which enables designing and prototyping of APIs in a simple manner. On the other hand, there is the *Open API initiative*[22], backed by Google, IBM and Microsoft among others. It supports the definition and establishment of a vendor independent format for describing REST APIs known as *Open API Specification (OAS)*, formerly named Swagger 2.0. Finally, the *OAS* currently holds a larger community of users and supporters than other candidates [35].

There is a nice open source framework called *Connexion* by Zalando which facilitates the development of microservices in *Python* following the API First approach. It is based on the well known Flask[23] microframework.

Architectural Decision #10

In the context of microservice library, facing simplicity, we decided for using Connexion to achieve an API first approach.

In terms of REST API design, the best practices defined the RESTful API guidelines by Zalando [40] as well as the RESTful API Design Quick Reference Card by Octo Technology [36] have been consulted. Further details on how much this affected the API design can be read in chapter 6.2.6.

4.5. Third-party Systems

The following third-party systems are in place (table 4.4). Together they create a system dependency. Therefore, it is vital to estimate how critical it would be if one system goes down.

[21] https://apiblueprint.org/
[22] https://www.openapis.org/
[23] http://flask.pocoo.org/

Dependency	Analysis & Rating
Apple	Kort Native is available for iOS thus creating a dependency. Also, the app is distributed via Apple's App Store. The dependency is rated marginal.
Google	Multiple dependencies exist. First, Kort Native is available for Android. Second, Google is used as an OAuth provider. The dependency is rated critical.
OSM	Since Kort is based on OSM, this system is rated critical. All the data for the mission derives from OSM. Finally, OSM is an OAuth provider.
KeepRight	One of the error sources is KeepRight. However, the dependency is rated marginal, since there is another error source which can be extended.
Overpass	The other error source is Overpass. The dependency is also rated marginal, however, since Overpass could be easily extended, it would be a pity to loose this option.

Table 4.4.: Third-party Systems

5. Design

This chapter covers everything related to design, be it design decisions, patterns or gamification elements.

5.1. Old Implementations

There have been previous implementations of the Kort game [11, 22, 26]. However, in scope of this project no working live version could be tested (since none existed). Of course there are screenshots of old implementations which give a basic idea of how these implementations behaved. For instance, features like *sneak peek* [11] for missions derived from the idea of presenting missions which are not around the player's position. In the meantime, however, given the new requirements, this feature is obsolete: By solving a certain amount of missions, players are allowed to solve any type of mission no matter how far away it is.

Since not only requirements but also technology has changed, it was decided to start off with a clean slate. By not having old ideas in mind, it enables to tackle design challenges with no preconception.

5.2. Design Prototypes & UI-Design Patterns

The design process started off with simple paper prototypes. They are easy and quick to draw. Next, wireframes mockups have been drawn. They convey the idea of a concrete design by having pixel perfect elements. Some of them will be presented in this chapter, along with some notes. For a complete extract of all designs, see the corresponding chapter 11 in the appendix.

5.2.1. Navigation

One of the most recurring design challenges is navigation. How should the user of the app be able to switch back and forth? How can he or she find specific views in an simple and quick manner? There are multiple solutions for this problem. Nowadays, many apps offer a so-called hamburger menu or sidebar menu. It offers a lot of space for all navigation categories. While this is great, this pattern also has some downsides [9]:

- Low Discoverability: The sidebar menu is hidden by default, that is why users need to identify the hamburger button as actionable.

- Less Efficient: It introduces navigation friction since it forces people to open the menu first and only then allowing to jump to the desired subsection.
- Clash with Platform Navigation Patterns: On iOS the upper left button is reserved for the back button in a view hierarchy since no hardware back button exists.
- Not Glanceable: If one category should be highlighted by a badge, people only notice when opening the side menu.

In contrast, using a tab bar the items are always visible, state can be directly presented in the according tab and there are no navigation gesture conflicts. With the help of A/B testing, an interesting study shows clearly that people are more likely to click on tabs than on side menus [34].

UI-Design Pattern: Navigation Tabs

Problem: Content needs to be separated into sections and accessed using a flat navigation structure that gives a clear indication of current location.
Solution: Each section or category is represented by a clickable tab button arranged in a tab bar.

Navigation tabs provide a clear visual indication of what content can be found and places the current location in context by highlighting it [29].

5.2.2. Present Mission Details

Since the map should receive as much space as possible, the question arises how to present mission details. It is clear to have mission annotations on the map and on click on a certain mission, details should be revealed. There are multiple options to do this.

One approach would be to click on the annotation and then open a new view in the hierarchy. This pattern is called *Breadcrumbs* and allows to browse back one or more levels. The downside of this approach is a possible disorientation on the user's side. Also, it takes some time to load or transition into the new view.

A better approach would be to use a helper view that slides in from the side and reveals more context in doing so. In case of Kort, it makes sense to have such a view popping in from the bottom when clicking on a map annotation.

UI-Design Pattern: Bottom Drawer/Sheets

Problem: Detail context information does not fit the current view.
Solution: Bottom sheets slide up from the bottom of the screen to reveal more content as a result of a user-initiated action.

The following figures depict each solution as wireframes.

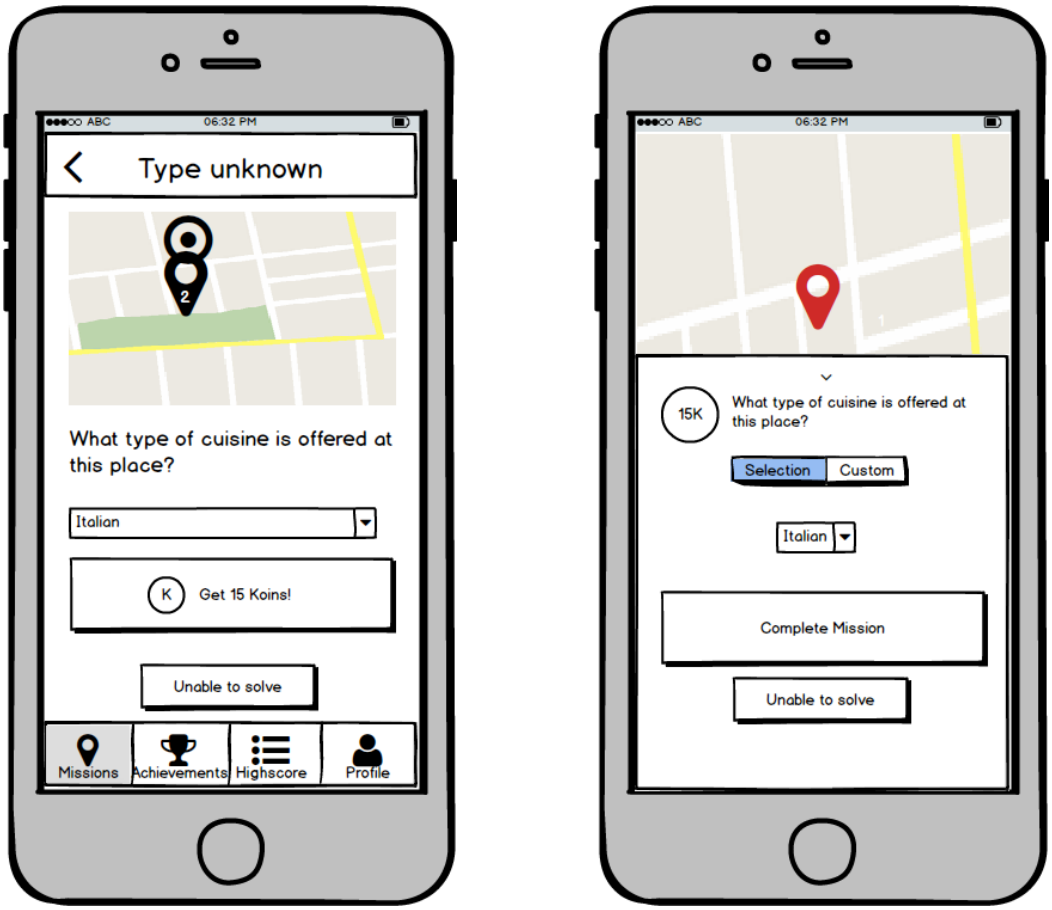

Figure 5.1.: UI-Design Pattern: Breadcrumb vs. Bottom Drawer

A welcome side effect of the Bottom Drawer pattern is the fact that the map does not need to be rerendered when just sliding in a helper view.

5.2.3. Update Content

There are multiple places in the app that have updateable content, be it achievements, highscores or even user updates. Again, there a multiple ways to support such a feature. One would be to have a separate button in the navigation bar, which initiates a refresh. The downside is a missing visualization for the update process. Also, the button itself consumes some space in the navigation bar.

On the other hand, there is a pattern that does not need any extra space for buttons. It's called *Pull to refresh* and brings along many advantages [30]:

- It is convenient: Users may update the screen whenever they want.
- It feels natural: The pull-to-refresh pattern has become a standard in mobile applications. Nowadays, users expect this feature.

- It saves bandwidth: Not to be neglected in a mobile environment.

UI-Design Pattern: Pull to refresh

Problem: The user wants to to retrieve more data or refresh already available contents on the screen.
Solution: As the user pulls down on the screen with a finger, visual feedback (refresh indicator) appears at the top of the list showing a progress of content update. If the user releases before reaching the refresh threshold, the refresh aborts and visual feedback disappears.

Figure 5.2 shows *Pull to refresh* in action as implemented in the Kort app.

Figure 5.2.: UI-Design Patttern: Pull to Refresh

Since the map view is pannable, this pattern could not be applied. Therefore, there is a button in the navigation bar which allows downloading new missions.

5.2.4. Further Considerations

From an early point on, it was decided to give the map enough space, due to its pivotal role of the game. Since the tab and navigation bar allocate quite some space, there needs to be a way of having a full screen map view.

A natural way to do this, is by tapping the map canvas. This way the same gesture can be used to switch between full screen and normal view. Also, the player does not necessarily need to switch to other tabs when solving missions.

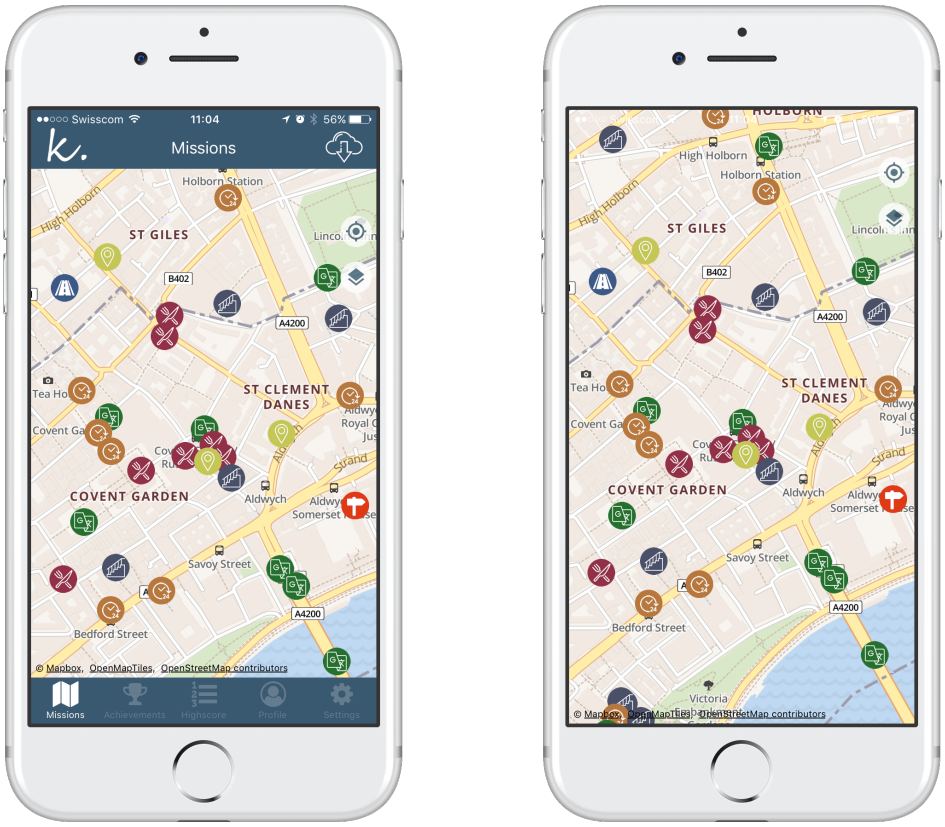

Figure 5.3.: Map Design: Normal and Full Screen mode

When selecting a mission on the map (map annotation), the bottom drawer slides in, indicating how much Koins there are to gain. If the player chooses to solve this mission, he or she can slide the drawer up and is presented with a corresponding input mask. Sometimes this is just a selection, a regular input text field or complex layouts such as opening hours. Figure 5.4 depicts the process of sliding up a mission.

The first time the app is started, a showcase is presented as depicted in figure 5.5. The main goal is to present what the app does in an illustrative way. The single features are separated in paginated views thus showing the progress in the showcase itself.

One important view worth mentioning is the players's profile. Figure 5.6 depicts the evolution from paper prototype to wireframe to actual implementation. It is the only place which presents the player's Koin and mission count. Since these values are elementary to the game, a separate view has been provided, rather than placing the values somewhere on the map.

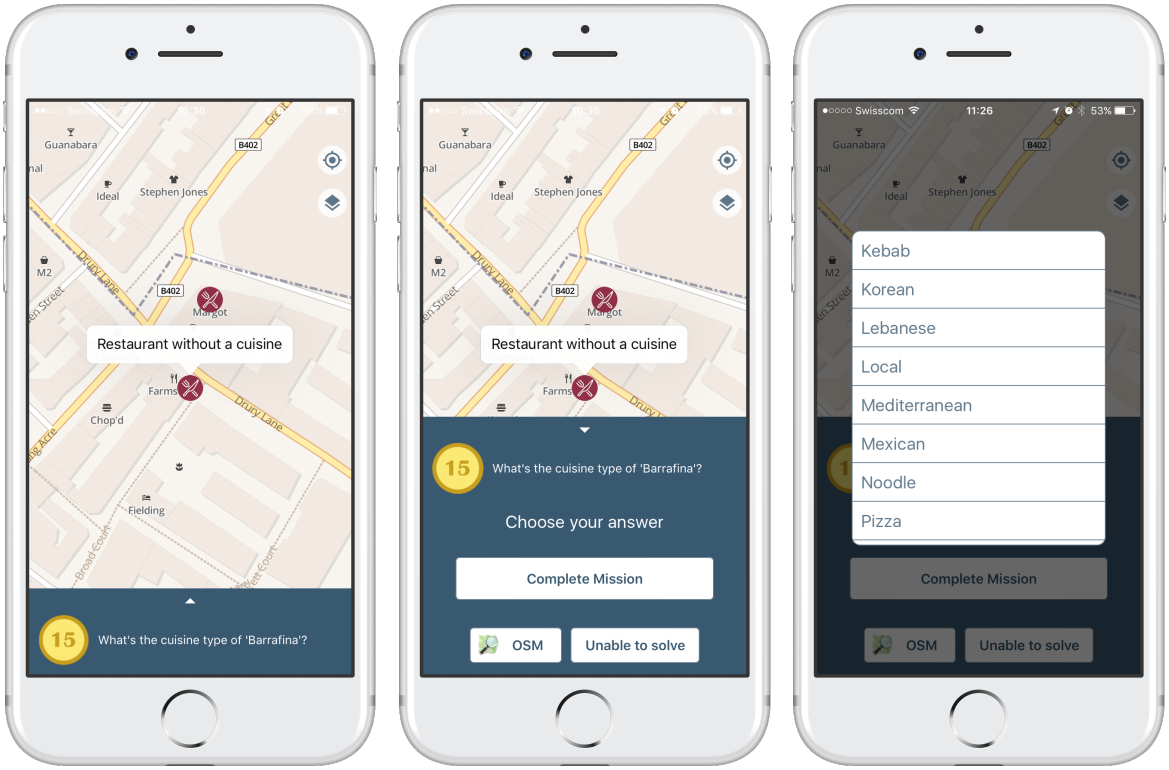

Figure 5.4.: Solving a Mission

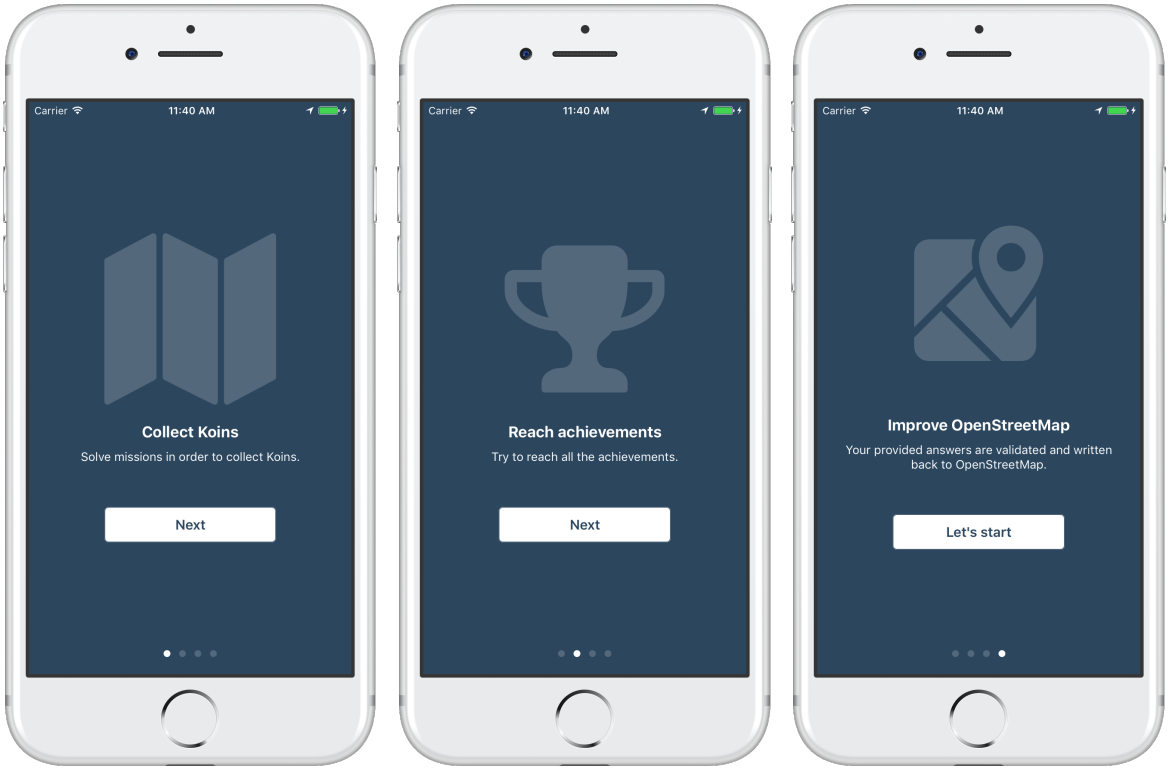

Figure 5.5.: Showcase

Figure 5.6.: User Profile: From Wireframe to Actual Implementation

5.3. Gamification

5.3.1. Basics

In order to identify which gamification elements make sense, it is vital to know the type of potential player. This will make it easier to design the experience that will drive player's behavior in a desired way. Fundamental work in this field has been conducted by Richard Bartle [10]. In his seminal work, he identified four types of players by studying players of MMOGs:

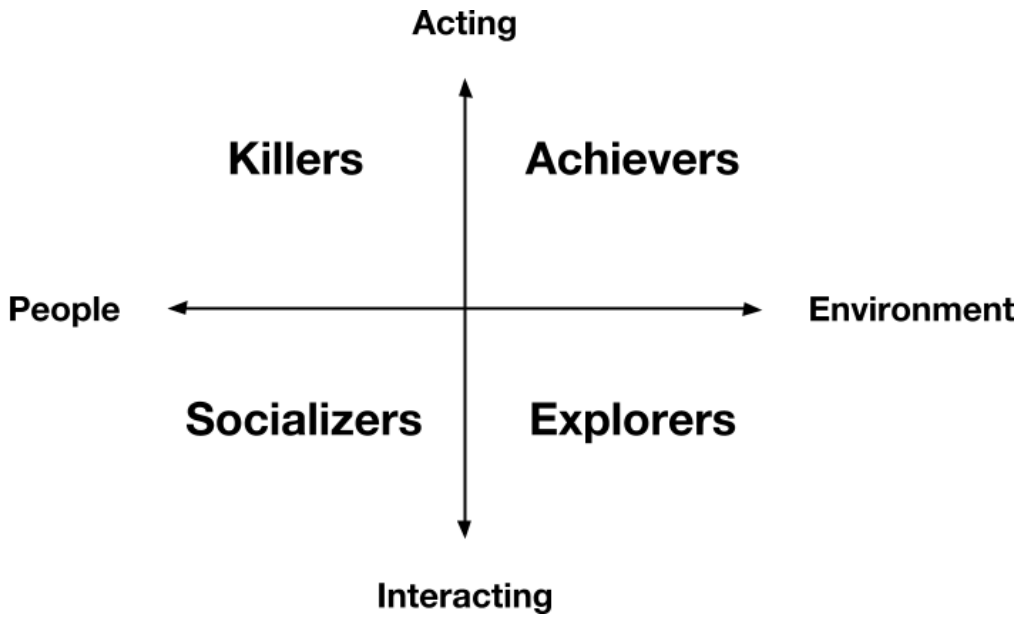

Figure 5.7.: Bartle's player types

Explorers

For the explorer, the experience is the objective. He or she likes to go out into the world and bring things back. These things may not always be easy to discover. The explorer is motivated by obfuscated achievements.

Achievers

Players who like to achieve things are an integral part of any game. Their focus lies on attaining status and achieving preset goals, preferably completely. Achivers are motivated by achievements.

Socializers

This player type's motivation lies in the benefit of social interaction. They do like winning, however, the game is a gateway for meaningful long-term interaction [17]. A vast majority of people fall into this category. They are motivated by newsfeeds, friends lists and chats.

Killers

Killers make up the smallest population of player types. Nevertheless, they play an important role. Unlike achievers, winning is not enough. As competitive as they are, they must win and someone else must lose. Finally, they want as many players to see the 'kill' hence gaining respect. They are motivated by leaderboards and ranks.

Of course, these types are not mutually exclusive. So a game's player type can have characteristics of all player types.

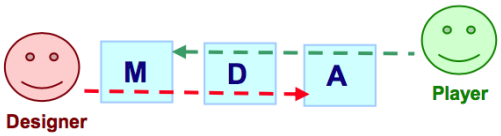

Figure 5.8.: MDA Framework [33]

Similar to Bartle's taxonomy, game designer Marc LeBlanc proposed a list of primary game pleasures:

1. **Sensation**: Everything involving one's senses. Seeing something beautiful, hearing music, touching, smelling and tasting are pleasures of sensation.
2. **Fantasy**: Describes the imagination of something, a world or a person which the player whishes to be.
3. **Narrative**: It does not necessarily mean story, but rather the dramatic unfolding of a sequence of events [21].
4. **Challenge**: Like an obstacle course, the game itself is a problem to be solved.
5. **Fellowship**: Everything social, ranging from friendship, cooperation to community is part of fellowship.
6. **Discovery**: A key game pleasure: While discovering new things such as places in a virtual world, a secret feature or strategy, the player finds himself in uncharted territory.
7. **Expression**: The pleasure of expressing onself includes things like editing the personal avatar or even building new worlds or levels which can be shared.
8. **Submission**: This describes the pleasure of entering the newly created world and coming back for more.

Those eight types desribe the Aesthetics or the 'fun part' in the MDA framework (Mechanics, Dynamics, Aesthetics) [33], whereas Mechanics describe the 'rules' (data representation and algorithms) and Dynamics describe the 'system' (run-time behavior of mechanics on player's input). The framework uses these definitions to demonstrate the motivating and deterrent properties of different dynamics. From a game designer's perspective, the mechanics gener-

ate dynamics which generate aesthetics. Since the designers are only able to influence the mechanics, which themselves produce meaningful dynamics and aesthetics for the player, this poses a challenge. From a player's perspective it is the other way around. Only through the aesthetics can they experience the game. And these are provided by the dynamics, which themselves emerge from the mechanics (see fig 5.8).

Most of Bartle's player types can easily be mapped to LeBlanc's aesthetics.

5.3.2. Applied Gamification Elements

A kort player is a mix of all those player types above. He or she is a

- *Achiever* because he or she likes to collect achievements.
- *Socializer* because he or she likes to share their achievements via social media (this feature is not yet developed).
- *Explorer* because he or she tries to find different types of missions in order to 'bring back' those trophies as achievements.
- *Killer* because he or she wants to show off their score in the leaderboards, competing against other players.

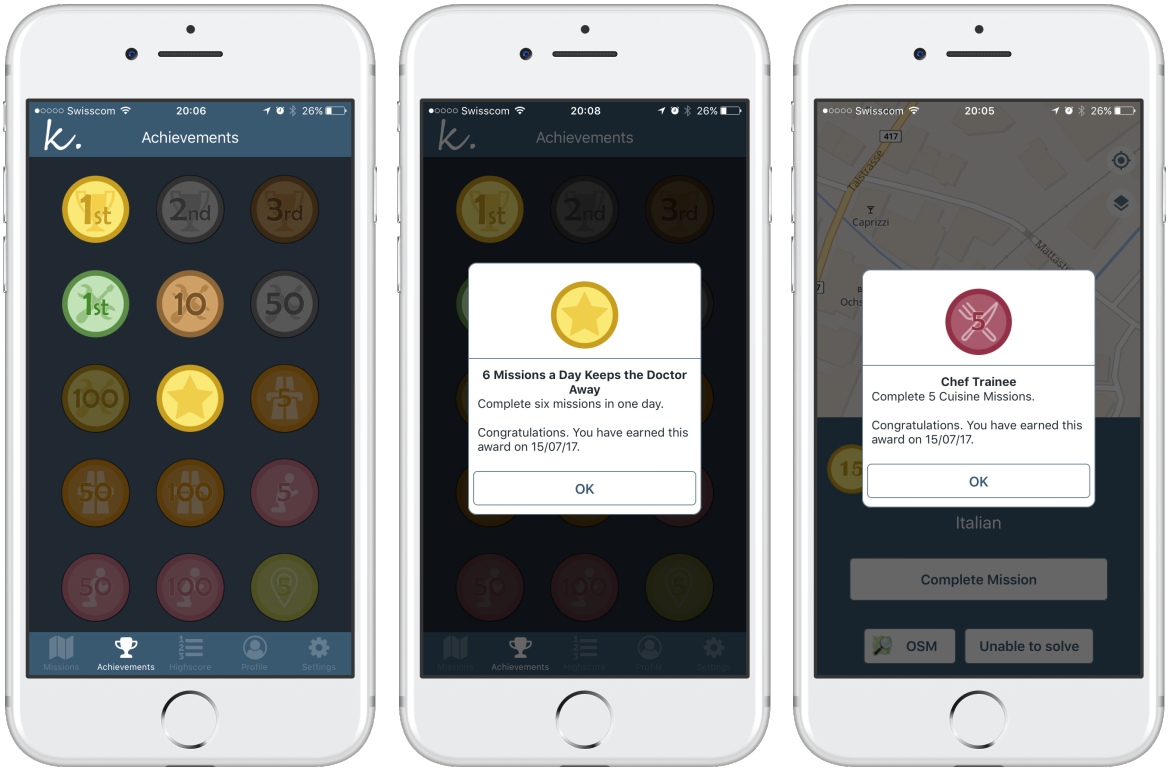

Figure 5.9.: Achievements - Trophy Case

The gamification elements already formulated do work well for this kind of game. However,

the contemplated elements can be improved which will be outlined in the following:

Achievements

Achievements play a pivotal role in Kort. It is popular to issue achievements in form of so-called badges. That is also the case in Kort, but they are not called badges, but rather achievements. There are many new achievement types which players can obtain, for instance one for completing six missions in one day, adding an additional dimension - time. Other new achievements include type-based achievements: For each mission type several achievements can be obtained. This should lead to a broader range of solved missions.

Winning such a trophy is of course less meaningful if one does not see the ones already won and the ones which are yet to be won. That is why there is such a trophy case in form of an achievement section (see figure 5.9). In order to unlock the power of those motivating achievements, the ones which have not been earned yet are greyed out. In contrast, the ones already won are bright. By clicking on a won achievement a playful animation honors the performance of the player.

Leaderboards

Leaderboards allow to make easy comparisons between players. They are easy to read since usually they are ordered lists with a score and a username (see figure 5.10).

As an extension to a regular leaderboard, a temporal dimension has been added. Now there are multiple leaderboards, looking back one day, one week, one month and a global one.

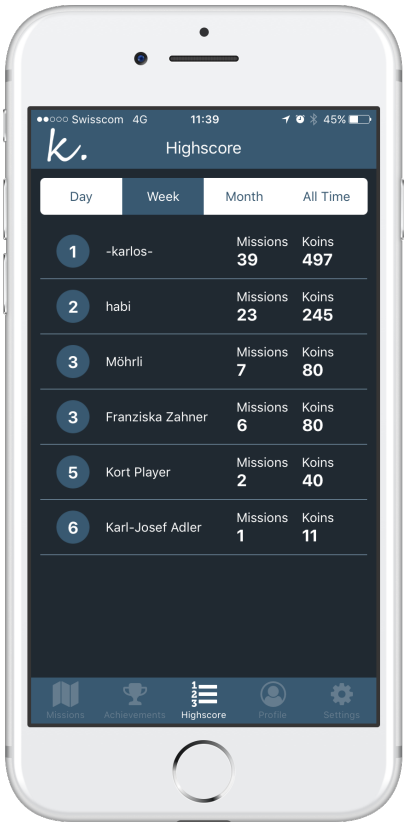

Figure 5.10.: Leaderboards

Note that in figure 5.10 the rank 3 is given twice since both players have the same amount of Koins (see chapter 6.2.5 for implementation details on this).

One could also think of adding more leaderboards, e.g. ones including only selected people, for instance just friends or people nearby.

5.3.3. Additional Elements

Gamification is not an exact science, but rather an iterative process in finding the right mixture of elements in order to create an engaging experience for the player. Figure 5.11 shows a multitude of gamification elements one can choose from.

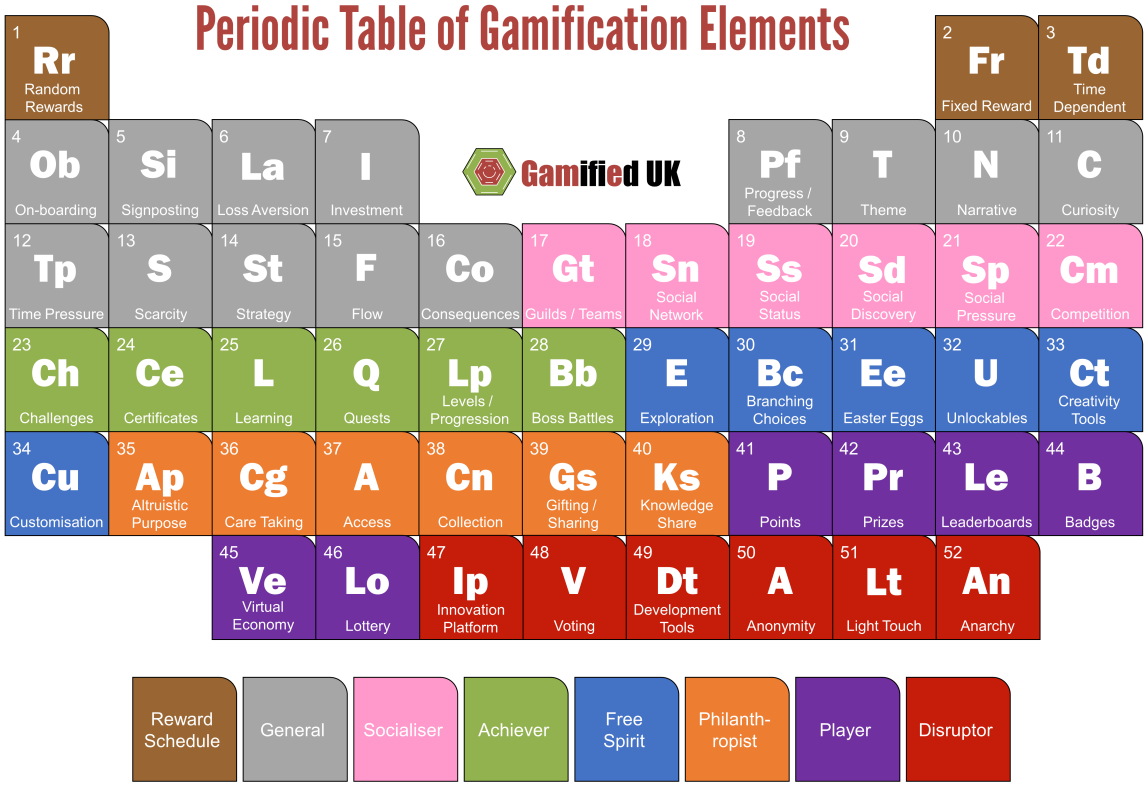

Figure 5.11.: Periodic Table of Gamification [25]

Of course it does not make any sense to try and combine all elements, but rather a wise choice of elements should be considered. This can only be achieved when the game designer tries to understand the type of player for his or her game with the help of Bartle's (see figure 5.7) and LeBlanc's taxonomies.

6. Implementation

This chapter covers implementation details worth mentioning. Again, it is structured into frontend and backend. Finally, known issues with the current implementations are listed.

6.1. Frontend

Here, we highlight several interesting implementation details on the frontend which deserve to get discussed separately.

6.1.1. React Native

As already mentioned in chapter 2.1.3, React Native means developing true native apps rather than webviews as a container for mobile websites. In order to visualize this, a view hierachy can be captured for instance in XCode.

Figure 6.1.: React Native View Hierarchy

Figure 6.1 depicts the view hierarchy of the achievements view along with wireframes. It can easily be validated that these views transform to native views. Specifically, on iOS the actual tree of views is composed of *RCTViews* which are a sublass of *UIViews*.

As a side note, it is revealed how achievements which are not obtained are greyed out: By overlaying a semi-transparent grey view matching the shape of the underlying view.

6.1.2. Redux

As already highlighted in chapter 4.3.2, Redux uses **action creators** and **reducers** for handling state. The following table lists all action creator files along with their reducers and a description about what they do.

Action Creators Reducer	**Description**
AchievementsActions AchievementsReducer	There is an action creator for downloading achievements. It calls the appropriate method of the API object. In order to align the last row at the bottom in the achievements section, empty objects are added so that the sum of items equals 3.
AnswerSelectionActions AnswerReducer	Here are all action creators related to the answer, be it either a selected answer or a typed one. Ultimately, there is an action creator which calls the appropriate method of the API object which handles sending the solution along with error handling. Finally, the state and content of a possible dialog box is handled.
AuthActions AuthReducer	Here is everything related to authentication: Handling of successful, as well as erroneous login or logout attempts, parsing deep links, verify Google token ids and finally setting the state of possible error dialog boxes.
HighscoreActions HighscoreReducer	There is an action creator for downloading the highscore list for the current tab. Also, when changing tabs an appropriate action creator is called.
MapActions MapReducer	Here one can find everything related to map actions: Switching full screen back and forth, updating of map center coordinates or location coordinates and whether the location accuracy is insufficient.
MissionActions MissionsReducer	Everything related to missions includes initiating the download of new missions at given coordinates, initiating the download of mission geometries as well as handling the state of whether a mission panel is visible.

NavigationActions NavigationReducer	Here one finds everything related to navgiation. There are action creators for clicking on a left or right button in the navigation bar. Also, there are action creators for force updating certain views, e.g. when new content is available. This way it is ensured that e.g. the achievements view is up-to-date after a successful mission, not only when a pull-to-refresh is initiated.
OpeningHoursActions OpeningHoursReducer	Since setting opening hours needs a lot of different states, a separate set of action creators has been created. They are responsible for adding/removing time entries, day entries, manual edits and the state of the different modals. The whole table state is represented in the reducer.
SettingsActions SettingsReducer	All action creators which are seen in the settings section are bundled here.

Table 6.1.: Action Creators & Reducers

Using the example of updating location coordinates, the whole workflow from action creator, action, dispatcher and reducer is illustrated in the following.

First, the action is defined. Actions are payloads of information that send data from the application to the store. For instance, a possible action looks like this:

```
export const LOCATION_UPDATE = 'location_update';
{
    type: LOCATION_UPDATE,
    payload: { latitude: 47.2232 , longitude:  8.8185 }
}
```

Code Snippet 6.1: Redux: Action

Second, an action creator is defined. They are simple functions that create actions. In our case, it returns the action from above.

```
export const locationUpdate = (location) => {
    return {
        type: LOCATION_UPDATE,
        payload: location
    };
};
```

Code Snippet 6.2: Redux: Action Creator

Actions describe that something happened. However, they don't specify how the application state changes in respone. This is done by reducers. They are pure functions, meaning that they take the previous state and an action, and return the next state. It is vital they stay

pure. This means no mutations, no API or calls of non-pure functions are allowed. The appropriate *MapReducer* looks as follows:

```
1  const INITIAL_STATE = {
2      currentLocation: { latitude: 0, longitude: 0 },
3      ...
4      ...
5  };
6
7  export default (state = INITIAL_STATE, action) => {
8      switch (action.type) {
9          case LOCATION_UPDATE:
10             return { ...state, currentLocation: action.payload };
11         ...
12         ...
13         default:
14             return state;
15     }
16 };
```

Code Snippet 6.3: Redux: Reducer

Note that the ES6 syntax for default arguments is used as well as the spread operator ... on line 10. The latter allows to copy field names and their values from a provided object onto the newly created object. Finally, the default case returns the previous state, which is important.

In order to improve readability and split the management of different states, reducer composition has been applied. It is a fundamental pattern of building Redux apps. Specifically, it looks like this.

```
1  export default combineReducers({
2      mapReducer: MapReducer,
3      answerReducer: AnswerReducer,
4      ...
5      ...
6  });
```

Code Snippet 6.4: Redux: Combine Reducers

Note that each of these reducers is managing its own part of the global state. The state parameter is different for every reducer and corresponds to the part of the state it manages.

Last but not least, a store provider has to be registered. This is usually done in the root component of the application. In our case this is in *App.js* and the render method looks like this:

```
1  import { Provider } from 'react-redux';
2  import { createStore, applyMiddleware } from 'redux';
3  import ReduxThunk from 'redux-thunk';
```

```
4 import reducers from './reducers';
5 ...
6 ...
7 render() {
8         const store = createStore(reducers, {}, applyMiddleware(
              ReduxThunk));
9
10        return (
11            <Provider store={store}>
12                <Router />
13            </Provider>
14        );
15    }
```

Code Snippet 6.5: Redux: Dispatcher

Note that for async actions, a redux middleware called *ReduxThunk* has been used. *Router* is the root component depending on some sort of state.

In order to connect to the store, *Redux* offers a special *connect()* function, which provides useful optimizations. The needed actions are connected and can then be used via props. This is done in the *GeoLocation* component which receives new location updates of the device.

```
1 import { connect } from 'react-redux';
2 import { locationUpdate } from '../actions/MapActions';
3 ...
4 ...
5 componentDidMount() {
6     navigator.geolocation.getCurrentPosition(
7       (position) => {
8         this.props.locationUpdate(position);
9         ...
10      }
11 }
12 ...
13 ...
14 export default connect(null, { locationUpdate })(GeoLocation);
```

Code Snippet 6.6: Redux: Using Actions Creators

In order to subscribe to changes of that state like the *Map* component does, one can do this as follows:

```
1 const mapStateToProps = ({ mapReducer }) => {
2     const { currentLocation } = mapReducer;
3     return { currentLocation };
4 };
5 ...
6 ...
7 export default connect(mapStateToProps, {})(Map);
```

Code Snippet 6.7: Redux: Receiving State Changes

Now, the object *currentLocation* can be accessed via props. When the value is changed and used in a *render()* method, React Native automatically re-renders the impacted part of the component.

6.1.3. Platform Specific Adjustments

Here we present parts which had to be adapted separately on each platform, iOS and Android.

URL Schemes - Deep Linking

As previously mentioned, in order to enable deep linking, certain platform specific adjustments needed to be made.

On Android this means to add the following *<intent-filter>* to the main activity in the *AndroidManifest.xml* file.

```
<intent-filter>
    <action android:name="android.intent.action.VIEW" />
    <category android:name="android.intent.category.DEFAULT" />
    <category android:name="android.intent.category.BROWSABLE" />
    <data android:scheme="kortapp" android:host="payload" />
</intent-filter>
```

Code Snippet 6.8: Deep Linking on Android

On iOS on the other hand, several adjustments needed to be made. First the following URL schemes need to be registered in the *Info.plist* file.

```
<key>CFBundleURLTypes</key>
  <array>
    <dict>
      <key>CFBundleTypeRole</key>
      <string>Editor</string>
      <key>CFBundleURLName</key>
      <string>com.googleusercontent.apps.UNIQUE_ID</string>
      <key>CFBundleURLSchemes</key>
      <array>
        <string>com.googleusercontent.apps.UNIQUE_ID</string>
      </array>
    </dict>
    <dict>
      <key>CFBundleTypeRole</key>
      <string>Editor</string>
      <key>CFBundleURLName</key>
      <string>kortapp</string>
      <key>CFBundleURLSchemes</key>
      <array>
        <string>kortapp</string>
```

```
21        </array>
22      </dict>
23    </array>
```

Code Snippet 6.9: Deep Linking on iOS #1

Note that UNIQUE_ID on line 7 and 10 is the client ID registered for this OAuth account.

Second, the following method in the *AppDelegate.m* file needs to be overridden.

```
1 - (BOOL)application:(UIApplication *)application openURL:(NSURL *)url
2   sourceApplication:(NSString *)sourceApplication annotation:(id)
        annotation {
3   if ([url.absoluteString.lowercaseString hasPrefix:@"kortapp://"]) {
4     return [RCTLinkingManager application:application openURL:url
          sourceApplication:sourceApplication annotation:annotation];
5   }
6   return [RNGoogleSignin application:application openURL:url
        sourceApplication:sourceApplication annotation:annotation];
7 }
```

Code Snippet 6.10: Deep Linking on iOS #2

If the URL has prefix kortapp, it is rerouted to the React Native Linking Manager. Otherwise the native Google library is called since these are all the deep links registered.

iOS: Insecure Connections

Even though the app communication is based solely on HTTPS, it is possible to use a local webserver for testing which is not necessarily HTTPS-based. Since iOS9, insecure connections are blocked, unless specifically allowed. This is also done in the *Info.plist* file. The appropriate key is called *NSAppTransportSecurity*. We would like to do without a detailed extract of the config file. If interested, refer to the appropriate file.

Third-party Libraries

Some third-party libraries needed platform specific adjustments. They shall not be disucssed in detail here. For the sake of completeness however, the affected libraries are listed here:

- react-native-google-signin
- react-native-i18n
- react-native-mapbox-gl
- react-native-vector-icons

Details on how to setup these libraries can be found in their respective documentation.

6.1.4. Map

As already mentioned in chapter 4.3.3, the map visualization was realized with the React Native Mapbox GL library. Unfortunately during the time of development there was no new release, 5.1.0 was the latest. This lead to problems with React Native 0.4.0 since it introduced breaking changes with iOS native headers[1]. The issue was resolved manually, by changing the native headers appropriately.

One topic which should be addressed at this point is full screen mode. The feature is considered important for this app and therefore needs a proper implementation. Unfortunately, the React Native Mapbox GL library does not offer a hook for catching double taps, because this gesture is reserved for zooming in. Hence, this behavior had to be cumbersomely replicated. So the goal is to allow double tapping for zooming as intended but single tapping for switching full screen mode. It is implemented as follows:

1. On each tap, store the current timestamp.
2. If the difference of the current timestamp with the last one is greater than a certain threshold, e.g. 250ms, toggle full screen mode.
3. Since the onTap() method gets also called when tapping on a map annotation, this circumstance needs to be handled appropriately.

This means that a full screen toggle can only be initiated when the used threshold is greater than the one measured when a double tap is initiated (somewhere around 200ms). This is not a nice solution, but maybe in a future release of the Mapbox GL library, there is better support for this.

6.1.5. Opening Hours

As previously mentioned in chapter 4.3.3, for entering time entries, the library *react-native-modal-datetime-picker* is used. It exposes a cross-platform interface for showing the native date and time pickers inside a modal. This approach was used, since entering date and time is so different on both platforms. On Android, there is the *TimePicker* component which looks and feels totally different compared with the iOS counterpart *UIDatePicker*. Finally, this library ensures a familiar usability on both platforms (see figure 6.2).

Note that when choosing to edit the generated date/time string, the view is not updated since it cannot map back the string to their respective fields. This feature is only intended for power users. In order to validate the input text, the opening_hours library is used (chapter 4.3.3). This ensures that no invalid date/time string gets sent. Consequently, it would be possible to send a string such as the following:

```
Apr-Sep: Mo-Fr 09:00-13:00,14:00-18:00; Apr-Sep: Sa 10:00-13:00
```

[1] https://github.com/facebook/react-native/releases/tag/v0.40.0

Figure 6.2.: Opening Hours: iOS vs Android Time Picker

6.1.6. Config

Finally, this subchapter goes into to configuration parameters which the frontend offers. Everything is included in two files: The first one is *SecretConfig.js*. This one is not on GitHub since it contains sensitive information. It includes the following key-value pairs:

Key	Value Description
`GOOGLE_IOS_CLIENT_ID`	the Google iOS client ID registered for this OAuth account
`GOOGLE_WEB_CLIENT_ID`	the Google web client ID registered for this OAuth account
`MAPBOX_ACCESS_TOKEN`	the Mapbox access token needed for using the Mapbox API

Table 6.2.: Secret Config

The second one is *Config.js* and contains the following key-value pairs:

Key	Value Description
MAPBOX_INITIAL_COORD_LATITUDE	the initial coordinate (latitude) to center the map on when there is no location service available, e.g 47.2232
MAPBOX_INITIAL_COORD_LONGITUDE	the initial coordinate (longitude) to center the map on when there is no location service available, e.g. 8.8185
MAPBOX_INITIAL_ZOOM_LEVEL	the initial zoom level for the map, e.g. 12
MAPBOX_MISSION_ZOOM_LEVEL	the zoom level when zooming to a mission, e.g. 17
MAPBOX_ANNOTATION_SIZE	the size of the mission map annotations, e.g. 30
MAPBOX_TAP_DELAY_IN_MS	the delay in ms for handling full screen mode
MAPBOX_DRAG_DELAY_IN_MS	the delay in ms needed for presenting the mission annotation popup by Mapbox (introduced because of wrong placement during 'zoomTo' phase)
API_VERSION	the current API version of the backend, e.g. v1.0
API_URL	the base URL to the backend, e.g. https://kort.dev.ifs.hsr.ch
DEEP_LINK_URL	the URL scheme to be used, e.g. kortapp://payload?
GOOGLE_VERIFY	the URL part for verifying Google IDs with the backend, e.g. '/google/verify'
OSM_LOGIN	the URL part for logging in via OSM, e.g. '/osm/login'
USER_INFO	the REST endpoint for fetching user information, '/v1.0/users'
MISSIONS	the REST endpoint for fetching missions, '/v1.0/missions'
HIGHSCORE	the REST endpoint for fetching highscores, '/v1.0/highscore'
ACHIEVEMENTS	the REST endpoint for fetching achievements, '/v1.0/achievements'
RADIUS_IN_M_FOR_MISSION_FETCHING	radius in meters to be considered when downloading missions, e.g. 5000
DISTANCE_DIFF_IN_M_FOR_MISSION_FETCHING	after how many meters from last location update new missions are downloaded, e.g. 1000
RADIUS_IN_M_FOR_MISSION_SOLVING	radius limitation in meters from current location for solving missions, e.g. 5000
NO_OF_MISSIONS_FOR_PERIMETER_CHECK	how many missions need to be solved until radius limitation is dropped, e.g. 20
ADDITIONAL_KOINS_FOR_STATS	how many additional koins to be earned when submitting statistics, e.g. 1

HIGHSCORE_LIMIT	number of highscore entries to be fetched, e.g. 10
KORT_GITHUB	the official GitHub repository of Kort Native
KORT_USERVOICE	the URL of the Kort the FAQ
KORT_WEBSITE	the URL of the Kort website

Table 6.3.: Secret Config

6.2. Backend

Here, we highlight several interesting implementation details on the backend which deserve to get discussed separately.

6.2.1. Old Backend

As already mentioned, the old backend consisted of shell, SQL, and PHP scripts and services. The goal was to completely replace the backend. However, since one of the error sources (KeepRight) was the same, it made sense to keep some of these shell scripts. It would have been a waste of time to re-engineer to whole workflow of downloading the database dump and copying it into the database. Additionally, to make it easier for migrating old data to the new database, some of the SQL scripts have been kept or slightly adapted. As a consequence, the database schema looks similar to the old one. Some parts have been stripped or extended.

To sum it up, the import scripts for KeepRight and the SQL scripts have been slightly adapted and put in a Docker container. This container is called *postgres*.

6.2.2. Mission Creator

In order to import new types of missions from Overpass, a new module has been created. It is called *mission creator* and is also part of the *postgres* Docker container. Basically, it is a python script which makes Overpass requests with the *Overpass QL* query language. For instace, if one wants to request all nodes on OSM within a given bounding box, which have a name tag, an amenity tag with key restaurant or fast_food, but a missing cuisine tag, one would state the following query:

```
node["name"~".*"]["amenity"~"restaurant|fast_food"]["cuisine"!~".*"](bbox);
```

All Overpass queries can be edited in the *overpass_ queries.py* file. Note that the bounding box has four floating point numbers separated by commas and should have the form (south,west,north,east). It is used to limit the results, since there may be an overwhelming number of records. Unfortunately, depending on the query and the area of the bounding box there may be errors from the server which need to be addressed. This circumstance lead to an approach of a small moving bounding box querying the same features over and over again like depicted in figure 6.3. While this approach works, it has one major downside. The smaller the bounding box, the longer it takes for querying the world. Parameters which work well in urban areas, meaning Overpass returns only little server exceptions, are as follows: Incrementing the latitude and longitude value by 0.1 resulting in a bounding box spanning an area of approximately 85km^2.

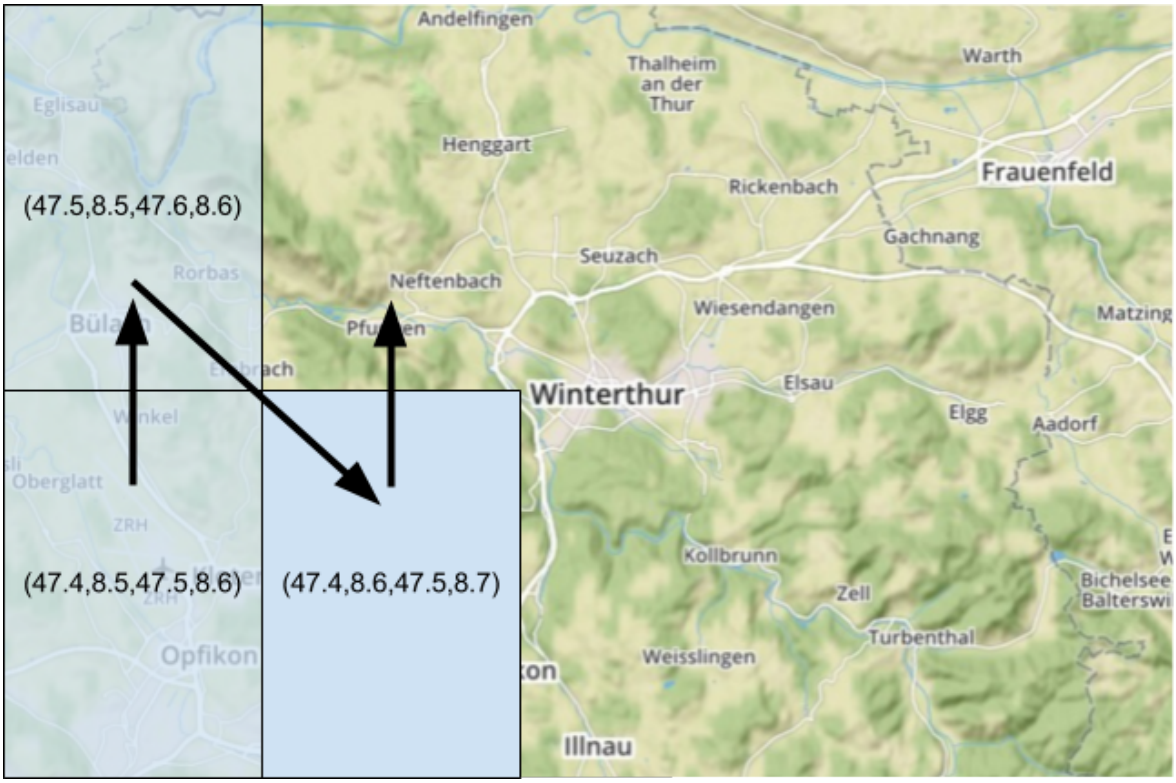

Figure 6.3.: Overpass Moving Bounding Box

A different approach would be to use dynamic bounding boxes as suggested on KeepRight[2]. However, it is not guaranteed that this would work better. The problems mentioned before still arise. One way to solve this problem is to completely gain control over Overpass by setting up an own instance.

Mission Creator writes a separate log file. See the following chapters for environment variables and run instructions.

6.2.3. Docker

The individual Docker containers have been introduced in chapter 4.4.1. Here we present ways to manage the containers as well as what environment variables there are. As previously mentioned, the Docker containers read from a local .env file. It looks as follows:

```
# Add Environment Variables
DEBUG=False
SECRET_KEY=<CHOOSE_YOUR_OWN_SECRET>
DB_NAME=osm_bugs
DB_USER=postgres
DB_PASS=postgres
DB_SERVICE=postgres
DB_PORT=5432
```

[2] https://www.keepright.at/planet.jpg

```
9  TOKENINFO_URL=tokeninfo:7979/tokeninfo
10
11 # Switzerland
12 BBOX_OVERPASS= 45.817315,5.701904,47.724545,10.524902
13
14 # OAuth Variables
15
16 GOOGLE_ID=<GOOGLE_OAUTH_ID_FOR_KORT_ACCOUNT>
17 GOOGLE_SECRET=<GOOGLE_OAUTH_SECRET_FOR_KORT_ACCOUNT>
18
19 OSM_CONSUMER_KEY=<OSM_OAUTH_KEY_FOR_KORT_ACCOUNT>
20 OSM_CONSUMER_SECRET=<OSM_OAUTH_SECRET_FOR_KORT_ACCOUNT>
```

Code Snippet 6.11: Docker .env file

The secret on line 3 is needed so that these variables are actually used. The bbox on line 12 is an example for Switzerland. The mission creator needs this variable for making Overpass requests over this area. The OAuth variables should be self-explanatory. There may also be a second file with environment variables which are server specific. This file is called *.envTest* and is intended for a test environment.

Now, some bash commands shall be given which hint at how these containers are started and used:

```
1  # setup/start postgres container and run given bbox on overpass, loads
       current KeepRight dump
2  docker-compose build && docker-compose up -d postgres
3
4  # setup/start API
5  docker-compose build && docker-compose up -d tokeninfo api nginx
6
7  # update data at a later date
8  # -k skip keepright update
9  # -o skip overpass update
10 docker exec -d kortcore_postgres_1 su -c "docker-entrypoint-initdb.d/
       update/update_db.sh -k" -s /bin/sh postgres
11
12 # read overpass logfile
13 docker exec kortcore_postgres_1 su -c "tail -f docker-entrypoint-initdb.d
       /mission_creator/overpass.log" -s /bin/sh postgres
```

Code Snippet 6.12: Starting Docker Containers

The first two commands are the only thing one needs to know in order to run Kort Core.

The update command can be executed manually or added to the crontab in order to have it run periodically.

6.2.4. OAuth

Figure 3.2 already revealed a glimpse on how the OAuth flow in Kort Core works.

There are many different OAuth flows depending on OAuth version (1.0, 2.0) and environments (web server, SPA, mobile apps among others). Following the KISS principle, it was decided to use a different workflow inspired by this site[3]:

1. User hits 'Login with Google'.
2. Google SDK talks to Google backend to get a token.
3. The frontend gives the backend this token.
4. The backend validates the token against Google's servers and receives user information.
5. The backend issues a self-generated authentication token (secret).
6. The frontend saves the backend's secret. From now on, the frontend only needs this secret token to communicate with the backend.

For each OAuth provider there are separate endpoints apart from the Kort API. At the moment, the following exist:

HTTP Verb	**Endpoint**	**Description**
GET	/google/login	Presents the login mask for Google login
GET	/google/login/authorized	Authorized redirect URI registered on Google developer console[4]. If successful, user information is fetched and persisted. The secret token is returned.
POST	/google/verify	The Google ID is verified. If successful, user information is fetched and persisted. The user ID and the secret token are returned. This additional endpoint is needed, so that mobile clients which themselves receive their Google ID can get hold of their kort user ID and secret.
GET	/osm/login	Presents the login mask for OSM login
GET	/osm/login/authorized	Authorized redirect URI registered on OSM OAuth page[5]. If successful, user information is fetched and persisted. The deep link URL is returned with the user ID and the secret as parameters.

Table 6.4.: Kort Core OAuth Endpoints

[3] https://library.launchkit.io/the-right-way-to-implement-facebook-login-in-a-mobile-app-57e2eca3648b
[4] https://console.developers.google.com
[5] https://www.openstreetmap.org/user/osmkort/oauth_clients

It was decided to only provide Google and OSM OAuth support. Facebook, which was supported in a previous implementation, did not have enough registrations with Kort thus not qualifying for development.

6.2.5. Highscore

The highscore or leaderboard is an essential gamification element (see chapter 5.3).

Technically, the API call depends on a SQL view. All used views are stored in the file *kort_views.sql*. Here is an example for a temporal leaderboard, e.g. for one day:

```
1  CREATE OR REPLACE VIEW kort.highscore_day AS
2  SELECT
3    ROW_NUMBER() OVER(ORDER BY koin_count DESC) AS id,
4    RANK() OVER(ORDER BY koin_count DESC) AS rank,
5    user_id,
6    username,
7    fix_count AS mission_count,
8    koin_count
9  FROM
10   (
11     SELECT
12       u.user_id,
13       u.username,
14       (SELECT SUM(fix_koin_count) AS count
15        FROM kort.fix f
16        WHERE (f.user_id = u.user_id AND f.create_date::DATE >=
              current_date)) AS koin_count,
17       (SELECT Count(1) AS count
18        FROM kort.fix f
19        WHERE (f.user_id = u.user_id AND f.create_date::DATE >=
              current_date)) AS fix_count
20     FROM kort.user u
21     WHERE (u.username IS NOT NULL)
22   ) AS score
23   WHERE koin_count IS NOT NULL
24   ORDER BY koin_count DESC
25 ;
```

Code Snippet 6.13: SQL View: Highscore

On line 3, `ROW_NUMBER()` is used to assign unique number to each row within the `PARTITION` given the `ORDER BY` clause. On line 4, `RANK()`, behaves like `ROW_NUMBER()` but rows with equal koin_count are ranked the same. This is needed for applying a 'standard competition ranking' strategy[6]. Figure 6.4 show this scenario. On line 16 and 19, the date of the solved mission is compared with the current date, resulting in having only results from today.

[6] https://en.wikipedia.org/wiki/Ranking#Strategies_for_assigning_rankings

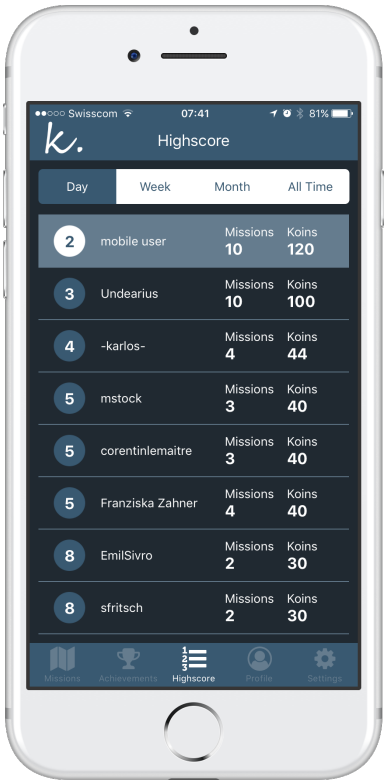

Figure 6.4.: Leaderboard: Standard Competition Ranking

6.2.6. REST API

The first step to build an effective microservice is to describe the resources that are going to be available in the API using the OpenAPI Specification. That means one considers what routes, parameters, payloads and which response codes the API produces. While designing the API, the RESTful API Design Quick Reference Card by Octo Technology [36] was consulted.

The following concepts have been followed:

- use of plural nouns instead of verbs
- the URL has mandatory versioning, meaning a new version of the API can be rolled out and apps which do not have the update yet still work
- use of HTTP verbs for CRUD-like operations
- I18n: support for different languages
- HTTPS support

The following is an extract from the *swagger.yaml* file which describes the API:

```
1 swagger: '2.0'
2 info:
```

```
  title: Kort API
  version: "1.0"
consumes:
  - application/json
produces:
  - application/json
basePath: /v1.0

paths:
  /missions/{schema_id}/{error_id}/solution:
    post:
      tags: [Mission]
      operationId: api.missions.put_mission_solution
      summary: submit solution to missions
      parameters:
        - $ref: '#/parameters/schema_id'
        - $ref: '#/parameters/error_id'
        - name: body
          in: body
          required: true
          schema:
            $ref: '#/definitions/missionSolution'
      responses:
        200:
          description: submit a mission solution, returns newly aquired
              achievements
          schema:
            type: array
            items:
              $ref: '#/definitions/Achievement'
        400:
          description: bad request
        404:
          description: an error occured
        500:
          description: internal server error

parameters:
  schema_id:
    name: schema_id
    description: schema id
    in: path
    type: string
    required: true
    pattern: "^[0-9]+$"
  error_id:
    name: error_id
    description: error id
    in: path
    type: integer
    required: true

definitions:
  Achievement:
```

```
56      type: object
57      properties:
58        achievementId:
59          type: integer
60          description: achievement id
61        achievementTitle:
62          type: string
63          description: achievement title
64        achievementDescription:
65          type: string
66          description: achievement description
67        achievementImageURI:
68          type: string
69          description: uri for the badge icon
70        achieved:
71          type: boolean
72          description: whether or not the achievement has been achieved
73        achievementDate:
74          type: string
75          format: date-time
76          description: the date on which the achievement has been achieved
```

Code Snippet 6.14: API YAML Description

From line 11 onwards, we define a route to `/missions/{schema_id}/{error_id}/solution`. Only POST requests are allowed. On line 15, the *operationId* tells which Python methods handles the call. On line 17, the parameters are defined. On line 25, the possible return responses are defined.

As already mentioned, one big advantage of this approach is having the documentation of the API always at hand. If one finds oneself lost with Swagger UI, we suggest to read this quite eye-opening part on Swagger[7]. It explains terminology and holds useful hints about how to work with Swagger.

For a complete documentation on the API, please refer to the appendix in chapter 12.

[7] http://idratherbewriting.com/2015/12/10/ten-realizations-using-swagger-and-swagger-ui/

6.3. Internationalization

Internationalization (I18n) is an important part of Kort since it allows to spread the product across the globe. In order to add as many translations as possible, the free plan of the *Transifex*[8] translation platform is used. With the help of the community, strings are translated little by little resulting in a complete translation of the ressources.

There are two ressource bundles, one for the frontend, the other one for the backend, both in their respective format. Transifex supports both, .json and .properties files which simplifies this workflow massively.

The properties for the frontend can be found at: `src/api/res/i18n/KortDB_LANG_SHORT.props`

The properties for the backend can be found at: `app/src/constants/i18n/LANG.json`

where LANG_SHORT represents the short language code, e.g. en, de, and LANG the language itself, e.g. English, German.

When adding new languages, *I18n.py* and *Translations.js* respecively must be extended accordingly.

The language of the app cannot be changed manually, but rather set automatically. The system language will be taken if available, otherwise English.

Currently, there are complete translations for English and German.

[8] https://www.transifex.com/odi/kort/dashboard/

6.4. Known Issues

There are some issues with the current implementation which could not be solved in scope if this project and shall be addressed here:

- On the map, the blue dot represents the current location provided by the GPS. There is also a blue circle around the dot, representing the accuracy of the signal. If this circle covers map annotations, the latter are not clickable anymore. This is quite annoying since it prevents solving missions. This might also be a bug of the React Native Mapbox GL library.
- On the map, map annotations represent mission at this place. Sometimes, it happens that there are multiple missions on the same place (e.g. opening hours and cuisine type). However, the player only sees one mission, since one annotation overlays the other. Because this scenario is likely to occur, this needs to be addressed.
- On one of the test devices (an old Galaxy S3 mini), mission annotations did not appear, but rather appear as black boxes. Actually, these images are available for different screen densities. The S3 mini should qualify for hdpi, however these do not appear nonetheless.

7. Results

In this chapter the resulting product is presented. Next, several possible future enhancements are discussed. Finally, it is topped off with a personal reflection.

7.1. Product Results

At the very beginning of the project it was decided not only to develop an all new frontend, but rather replace the existing backend as well. This resulted in several architectural decisions based on the requirements given.

First, in order to make the backend easier to maintain, it was clear to use some sort of containerization. That means that an application is encapsulated in a container with its own operating environment, getting rid of any dependencies as well as simplifying the deployment process. This architecture was realized with *Docker*[1]. Each component of the backend lives in its own container, e.g. PostgreSQL database and API.

Second, the process of adding new error sources (missing data) has been extended. The data gathering process now includes OSM errors not only from *KeepRight*[2], but also from *Overpass*[3]. By using the Overpass API it is now possible to easily add new mission types for a global dataset. In scope of this work, this extension has been applied for places with missing opening hours among others.

Third, the PHP API has been rewritten with easier to read *Python* code. It makes use of the *Connexion*[4] framework by *Zalando* which sits on top of *Flask* and exposes a RESTful API based on OpenAPI 2.0 Specification (FKA Swagger Spec) [4]. This allows a clear and up-to-date live documentation of the interface since Connexion works with a specification-first approach and not the other way around. As a huge bonus, the Swagger Web Console UI allows easy testing by calling the API's endpoint directly.

[1] https://www.docker.com/
[2] http://www.keepright.at/
[3] https://overpass-turbo.eu/
[4] https://github.com/zalando/connexion

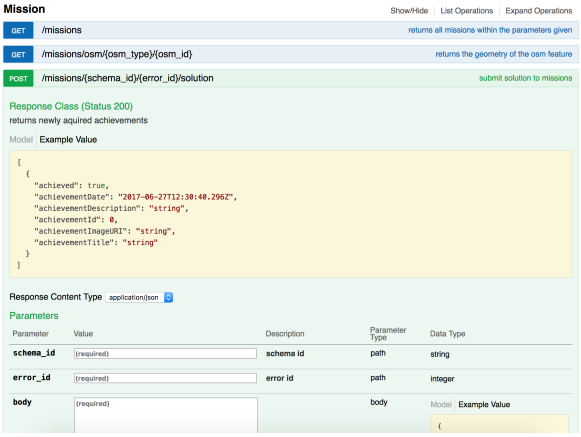

Figure 7.1.: Swagger API Documentation

Finally and most importantly, the frontend has been redesigned and rewritten with *React Native*. Now, there exists a native app for the Android and the iOS platform. Users can login with their Google or OSM account, which was an important requirement, since users do not like to create new accounts and remember new passwords. Behind the scenes, the API uses the OAuth standard. Nevertheless, it is also possible to discover the app without logging in.

The map features vector tiles by the *OpenMapTiles*[5] project. Compared to raster tiles, this allows reducing data transfers dramatically which is essential in mobile applications.

Furthermore, new mission types have been introduced, namely missing building level, missing cuisines for restaurants and missing opening hours for community facilities. An adapting input mask is presented for each mission type, e.g. for opening hours a native date/time picker is presented. Along with new missions, the number of achievements could also be increased. Now there exist achievements for each mission type, as well as a special achievement which can be unlocked when solving six missions in one day.

In order to increase the competition with other players, more leaderboards have been introduced. It is now possible to be the leader of the day, week or month.

A showcase featuring the possibilities is presented when the player uses the app the first time.

The work could be successfully presented at SotM-FR and received broad interest which resulted in many beta testers.

[5] https://openmaptiles.org/

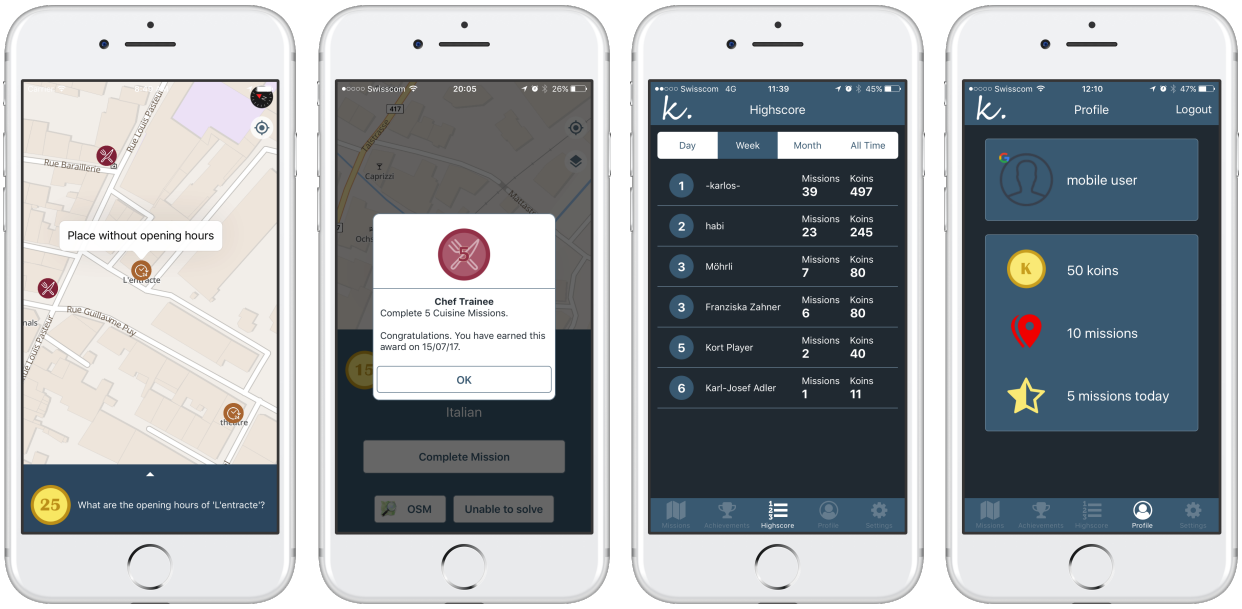

Figure 7.2.: Kort Native App

7.2. Outlook

The groundwork has been laid. The next goal is to bring the app to the App Store and Google Play Store. This will be happening shortly after a successful beta phase, which is still ongoing as of this writing.

After a successful launch later this summer, it is possible to add further enhancements, some of which shall be presented at this point.

First of all, it is vital to add a module to the backend which automatically writes back validated missions to OSM. This will strengthen the reputation of this game in the OSM community. This could be achieved by creating an additional Docker container responsible for that job. On an additional note, the mission creation/update process needs to be accelerated when requesting data from the Overpass API. One could also think of having a separate API to add missions sources. In a basic variant this could look as a way of adding Overpass queries with additional attributions. Before going live with the new Kort, the old data needs to get migrated. This is important because returning players wish to keep their previous accomplishments.

Second of all, the statistics gathered with iTunesConnect, Google Play Store and Mapbox telemetry[6] should be used to further improve existing parts of the app. App Store reviews should be addressed if needed.

Third of all, it is possible to add more spatial awareness, such as spatial achievements, leaderboards or promotions. The latter was available in the old Web app and was not a goal in scope of this work.

[6] https://www.mapbox.com/telemetry/

Fourth of all, the fact that this game is now a true Native App opens up the possibilities to have push notifications (e.g. for promotions) as well as Deep Linking from and to other apps such as Facebook, Whatsapp or mapping apps. It is also possible to create offline functions, for instance the ability to download the map for a whole region and sync the solved missions when online again.

Fifth of all, when the React Mapbox GL library comes out of its infancy, there is better support on both platforms for custom map annotations. This allows to have better performance and more layout options on map annotations. Furthermore, it may be possible to do a clustering on annotations thus hinting at hot spots of missions at any zoom level at all times.

Finally, additional gamification elements could be introduced such as experience points (XP) or levels [17]. Levels do not necessarily need to be numbers, but rather funny titles. These could then also be visible in the leaderboards.

7.3. Reflection

Implementing the backend with Python was a good decision. Even though not working with Python before, apart from writing some mini scripts, it offered me a compelling insight. Being able to realize a complex backend without writing too much boilerplate code is nifty. I especially like the API first approach which Connexion supports. This way there is always an up-to-date documentation at hand.

Working with React Native was a new experience. However, after a couple of sample projects I got to know the quirks of it and immediately started loving it. This is especially true for layouting. Flexbox solves design issues which are typically challenging on a mobile platform. It does it so well and for two platforms. As an iOS and Android developer I appreciate this circumstance very much.

However, it's not just designing. In my opinion, React Native's hidden awesome feature is hot reloading. React Native is able to re-render the current view in the iPhone Simulator without the need of recompiling the XCode project. This is a huge advantage and it saves a lot of development time in the long run. Because there are already so many libraries out there, I did not get the chance to write any native modules which would have been interesting. I am confident that React Native will claim its definite role in the mobile development world.

Part II.

Project Documentation

8. Tools

This chapter should give an overview on what tools have been used in scope of this work. These range from IDEs to graphics editors and miscellaneous.

8.1. IDEs

Since different programming languages have been used, many IDE's were applied.

8.1.1. PyCharm

PyCharm is a Python IDE by JetBrains[1]. It is a very nice choice for developing in Python as it provides useful features such as:

- Refactoring
- Debugger
- Support for web frameworks e.g. Flask
- Support for Docker
- Code Analysis
- Git Integration

8.1.2. Visual Studio Code

A very fine and versatile editor by Microsoft[2] which is completely free. It has been used for developing mainly with JavaScript. It also features GIT integration and many extensions, e.g. for React Native, ESLint, Docker or Latex.

[1] https://www.jetbrains.com/
[2] https://code.visualstudio.com/

8.1.3. XCode

The official IDE by Apple for developing Swift or Objective C for iOS.[3] It is used for developing, building and distributing iOS apps.

8.1.4. Android Studio

The official IDE by Google for developing Java for Android.[4] It is used for developing, building and distributing Android apps.

8.2. Graphics

Several graphic tools have been used.

8.2.1. Photoshop

Photoshop has been used for editing SVG graphics and transforming them to PNG. For instance, the achievement badges and mission annotations have been designed with Photoshop.

8.2.2. Density Converter

Since iOS and Android need raster graphics in different sizes, it is rather cumbersome to do this task manually. There is a great tool called *Density Converter*[5] which converts single or batches of images to Android, iOS, Windows and CSS specific formats and density versions given the source scale factor or target width/height in dp. Very helpful indeed.

8.2.3. frameit

For documentation purposes, it is nicer to have app screenshots with a frame representing a real device. There is a command line tool called frameit[6] which wonderfully transforms batches of images and frames them with the desired device frame.

8.2.4. Balsamiq Mockups

Balsamiq Mockups by Balsamiq Studios[7] is a graphical user interface mockup and website wireframe builder application. It allows arranging pre-built widgets using a drag-and-drop WYSIWYG editor.

[3] https://developer.apple.com/xcode/
[4] https://developer.android.com/studio/index.html
[5] https://github.com/patrickfav/density-converter
[6] https://github.com/fastlane/fastlane/tree/master/frameit
[7] https://balsamiq.com/

8.3. Miscellaneous

8.3.1. DataGrip

DataGrip is an IDE by JetBrains[8] aimed at developers working with SQL databases. It offers great support for all major database drivers. It also offers great visual data schema exports.

8.3.2. Paw

In order to test the API, one could use the Swagger UI. However, for always recurring requests, an API tool like Paw[9] comes in really handy. It also offers inspection of server responses, generation of client code and export of API definitions.

8.3.3. ngrok

When developing for mobile platforms one wants to test out a working prototype on a real device really fast. However, when the local backend needs to be available apart from the local network, further configurations are needed. This is where ngrok[10] steps in. It exposes a local server behind a NAT or firewall to the internet as easily as a simple command on the command line. This is an essential tool.

8.4. Continuous Integration

Travis CI was used for continuous integration[11]. So after each GIT commit, the Travis CI worker pulls the current master branch from GitHub and builds the app. In case of an error, an email is sent to the developer.

8.5. Testing

In the following, the different test frameworks or workflows are described both for the frontend as well as for the backend.

8.5.1. Backend

For testing Python applications there are several options. For Unit Tests, the *pytest*[12] framework was used.

[8] https://www.jetbrains.com/
[9] https://paw.cloud/
[10] https://ngrok.com/
[11] https://travis-ci.org/kort
[12] https://docs.pytest.org

In order to test the API itself, the *HTTPie*[13] client was used. It is a great CLI for testing RESTful APIs and compared to CURL it has an intuitive UI, JSON support and last but not least syntax highlighting.

The tests are run by Travis CI or locally by a simple

```
pytest
```

command.

For testing the API, the API must be available on the localhost. Then the test script can be exectued with:

```
sh tests/test_api.sh
```

Note that, if no localhost is available one could also point the URL to a working test environment such as https://kort.dev.ifs.hsr.ch/v1.0.

8.5.2. Frontend

For testing React Native apps there are multiple options. For Unit Tests, the *JEST*[14] framework was used. It is a good choice for testing JavaScript applications especially with React. An additional test type which JEST supports is Snapshot Testing (ST). An ST renders a UI component, takes a screenshot, then compares it to a reference image stored alongside the test. The tests are run by Travis CI or locally by a simple

```
npm test
```

command.

However, the most important tests are the ones with real devices. Only then, the true responsiveness of the app is revealed. Also realistic network conditions are in place. The following test devices have been used

- iPhone 5C with iOS8
- iPhone 6 with iOS10
- Samsung Galaxy S3 mini with Android 4.3
- Samsung Galaxy S5 with Android 6

Finally, there is a testing user group of over 20 people with vibrant feedback, which is invaluable.

[13] https://httpie.org/
[14] https://facebook.github.io/jest/

8.6. Coding Guidelines

In the backend, Python code was checked against the PEP8 Style Guide for Python Code[15].

In the frontend, ESLint[16] has been used for code linting. The configuration was taken from RallyCoding[17].

[15] https://www.python.org/dev/peps/pep-0008/
[16] http://eslint.org/
[17] https://www.npmjs.com/package/eslint-rallycoding

9. Installation

This chapter covers from scratch how the whole app is setup and distributed. On the opposite side, the installation of the backend is documented. Finally, it is shown how new mission types are added to the whole system.

9.1. Kort Native

9.1.1. Debug Mode

For running iOS applications XCode must be installed on a macOS machine. These are the steps for debug mode:

1. Clone the project `git clone https://github.com/kort/kort-native.git`
2. For Android run `react-native run-android`
3. For iOS run `react-native run-ios`

To enable remote debugging, shake the device.

Alternatively, one could also open the projects in folders *ios* and *android* with XCode and Android Studio respectively.

9.1.2. Release Mode

If the goal is to build a binary which can be distributed, the following commands need to be run:

For Android:

```
# generate a private signing key using keytool
keytool -genkey -v -keystore my-release-key.keystore -alias my-key-alias
    -keyalg RSA -keysize 2048 -validity 10000

# assemble and release
cd android && ./gradlew assembleRelease
```

Code Snippet 9.1: Release Android Version of Kort Native

For detailed description check the documentation[1]. The resulting APK is singed and can be distributed through a distribution platform like the Google Play Store.

For iOS, open the project in XCode and switch the scheme from 'debug' to 'release' (see figure 9.1)

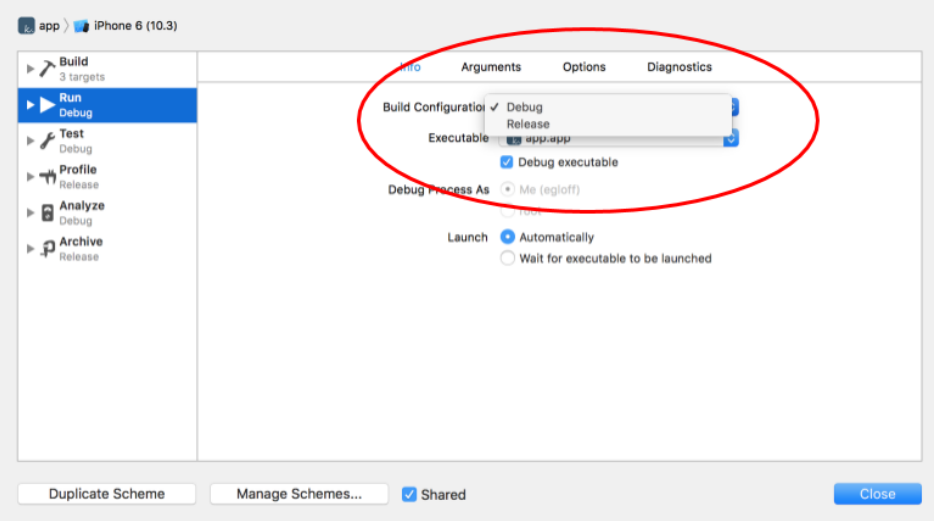

Figure 9.1.: iOS Release Mode Scheme

In order to distribute the iOS app there must be an appropriate provisioning profile. In XCode, choose *Product/Archive*. The binary is then available in the *Window/Organizer*. One can then choose to distribute the app via App Store (also the way for TestFlight) or export the binary for different ways of distribution.

The beta test app is distributed via *TestFlight*[2].

9.2. Kort Core

The following commands enable the use of the Kort Core backend. Make sure to create an .env file at the project root as described in chapter 6.2.3.

```
# clone the project from github
git clone https://github.com/kort/kort-core.git

# setup/start postgres container and run given bbox on overpass, loads current KeepRight dump
docker-compose build && docker-compose up -d postgres

# setup/start API
docker-compose build && docker-compose up -d tokeninfo api nginx
```

[1] http://facebook.github.io/react-native/releases/0.40/docs/signed-apk-android.html
[2] itunesconnect.apple.com

```
9
10 # FINISHED
11
12 # if one wants to update the mission database the following commands come
       in handy
13 # -k skip keepright update
14 # -o skip overpass update
15 docker exec -d kortcore_postgres_1 su -c "docker-entrypoint-initdb.d/
     update/update_db.sh -k" -s /bin/sh postgres
16
17 # read overpass logfile
18 docker exec kortcore_postgres_1 su -c "tail -f docker-entrypoint-initdb.d
     /mission_creator/overpass.log" -s /bin/sh postgres
```

Code Snippet 9.2: Setup of Kort Core

9.3. Adding New Mission Types

Currently the following mission types are available:

Mission Description	**Source**	**OSM Tag**
Motorway without reference	Keepright	ref
Object without a name	Keepright	name
Missing speed limit	Keepright	maxspeed
Type of track unknown	Keepright	tracktype
Place of worship without religion	Keepright	religion
Language of the name unknown	Keepright	name:XX
Street without a name	Keepright	name
Restaurant without a cuisine	Overpass	cuisine
Place without opening hours	Overpass	opening_hours
Missing Levels	Overpass	building:levels

Table 9.1.: Current Kort Missions

Adding new mission types needs work on both the frontend as well as the backend.

9.3.1. Backend

Several steps are required when adding new mission types:

1. Add a new entry to the table *kort.error_type* with a new *error_type_id* (kort_data.sql).

2. If additional achievement badges should be available, create the appropriate entries in this file as well.

3. Create an appropriate QL Overpass query as seen in chapter 6.2.2. Add also the error_type_id chosen in the step before (overpass_queries.py)

4. In the next import run these new missions should be added to the database and available to the client

5. Do not forget to add the appropriate language translations to the localization files

9.3.2. Frontend

In the frontend there are only minor tweaks necessary. In fact, only graphic files must be provided. There are two ways to accomplish this:

This can be done either by providing an URL to an image on the server in the REST services. As an example, let us consider the achievement endpoint in the Kort API. There is a field *achievementImageURI*. This usually points to a local image on the mobile app. This has two big advantages: First, load times are way faster. Second, device optimized graphics can be loaded (especially useful for high density screens). The corresponding field in the mission endpoint is called *image*. Now, one could also choose to provide an URL which points to a remote image.

Alternatively, graphic files can be provided with the native app binaries. As a consequence this approach requires an app update. However, if one chooses to go with local assets, it is highly recommended to use the Density Converter tool presented in chapter 8.2.2.

10. Project Management

10.1. Organization

This work has been done as a one-man operation supervised by Prof. Stefan Keller. The source code is open source on GitHub[1][2]. Additionally, the source code as well as all other files are provided on a separate USB stick.

10.2. Coordination

Coordination meetings have been held every two weeks either by telephone or in person. An agile software development process with adaptive planning as well as continuous improvements has been followed.

10.3. Monitoring

GitHub was used as a code revision control and source code management tool. A beta version v1.0-b1 has been released on June 26th.

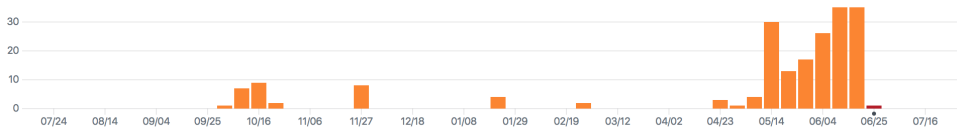

Figure 10.1.: Kort Core Commits

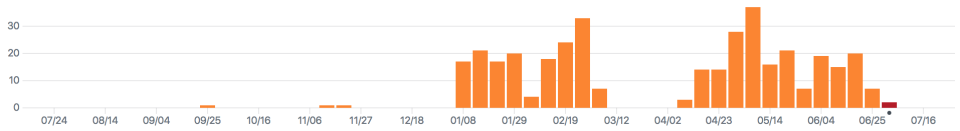

Figure 10.2.: Kort Native Commits

Figure 10.1 and 10.2 nicely depict the development process which was more intense at the end. At the beginning there was a lot of requirements engineering and technology analysis.

[1] https://github.com/kort/kort-native
[2] https://github.com/kort/kort-core

10.4. Target-Performance Comparison

Each week all the tasks were analyzed in a target-performance comparison sheet. The predefined milestones at the of a 4-week-sprint were almost always met.

Week	Sprint	Date	Tasks	Problems / Notes	Milestones	Actual Time	Target Time
39	1	19.09.16	Kickoff Meeting	React Native Workshop with Sebastian Deutsch // Getting started // Kickoff @ Liip		5	20
40	1	26.09.16		Evaluate REST services -> Flask Rest+ with SQLAlchemy and Swagger UI		10	20
41	1	03.10.16	Meeting Skype		Design whole application with all ist components and technologies used	17	20
42	1	10.10.16		Add instructions for website changes // new framework evaluated -> connexion	DB Design // Dockerizd Prototyp // Travis CI // on gitHub // Unit Test Framework	13	20
43	2	17.10.16	Meeting		first services	11	20
44	2	24.10.16		Tool for testing OAuth: paw // instructions for website changes delivered	OAuth integration in backend, first service: google	15	20
45	2	31.10.16	Meeting	evaluate PostgREST, ToroDB // schema validation on PUTs		1	20
46	2	07.11.16	Meeting		dummy missions	10	20
47	3	14.11.16				19	20
48	3	21.11.16	Meeting		Oauth : OSM service	19	20
49	3	28.11.16			React Native first steps	9	20
50	3	05.12.16	Meeting		mapbox integration & benchmark	6	20
51	4	12.12.16		BREAK	BREAK	0	20
52	4	19.12.16	Meeting	BREAK	BREAK	0	20
53	4	26.12.16				4	20
1	4	02.01.17			first map prototype // react and redux basics // Design Prototype	33	20
2	5	09.01.17	Meeting	RN 0.40 breaks current Mapbox RN library -> pull request pending		25	20
3	5	16.01.17		Google disallows webview Oauth -> Google Sinin button // no modal login view possible when redirecting to other Oauth modal views	webservice integration in app	25	20
4	5	23.01.17	Meeting			36	20
5	5	30.01.17		when are missions loaded, which radius which zoom level? Is it possible to solve missions when not within radius?		38	20
6	6	06.02.17	Meeting	Telemetry decisions / map styles		17	20
7	6	13.02.17		What happens if user answer is wrong? // always allow freetext? // selected annotation not supported // onDevice validation for input	Add Gamification elements	25	20
8	6	20.02.17				29	20
9	6	27.02.17	Meeting	Imprint? _> YES Zoom in / out buttons? -> NO / koin design ok? // settings? -> YES	fast prototype ready	24	20
10	7	06.03.17			Testing	6	20
11	7	13.03.17		VACATION	VACATION	0	0
12	7	20.03.17		VACATION	VACATION	0	0
13	7	27.03.17		VACATION	VACATION	0	0
14	8	03.04.17			adaptions from last meeting	10	20
15	8	10.04.17				6	20
16	8	17.04.17	Meeting			22	20
17	8	24.04.17				20	20
18	9	01.05.17	Meeting			34	20
19	9	08.05.17		keepright data only has points not whole geometries	mission creator , data by KeepRight, ideally with same shell script as used before -> direct access to db	43	20
20	9	15.05.17				36	20
21	9	22.05.17	Meeting			14	20
22	10	29.05.17			SOTM FR - LIGHTNING TALK	34	20
23	10	05.06.17				30	20
24	10	12.06.17	Meeting		Backend deployen // App ready for testing	36	20
25	10	19.06.17		Android Google SignIn problems in release mode WRONG SIGNIN -> added SHA1 of keystore to firebase console	Start documentation	42	20
26	11	26.06.17				37	20
27	11	03.07.17	Meeting			16	20
28	11	10.07.17				38	20
29	11	17.07.17	Meeting			42	20
30		24.07.17			Documentation finished	24	
31		31.07.17					
						881	820

Figure 10.3.: Target-Performance Comparison and Monitoring Sheet

Note that in the end the project took longer than planned (+7%).

Part III.

Appendix

11. Design Prototypes

The following wireframes have been created with Balsamiq[1].

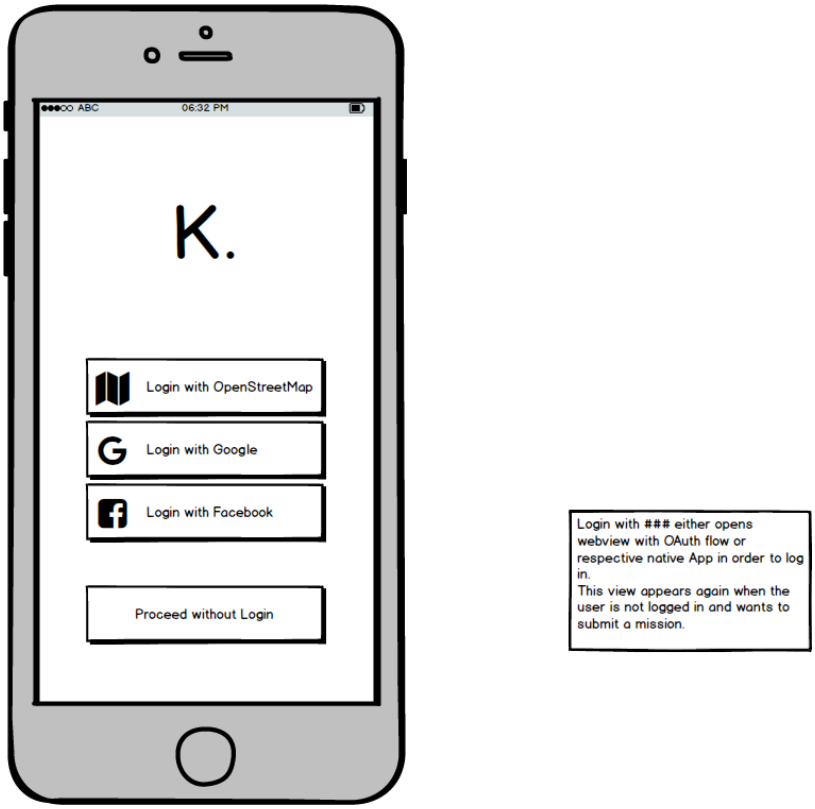

Figure 11.1.: Login

[1] https://balsamiq.com/

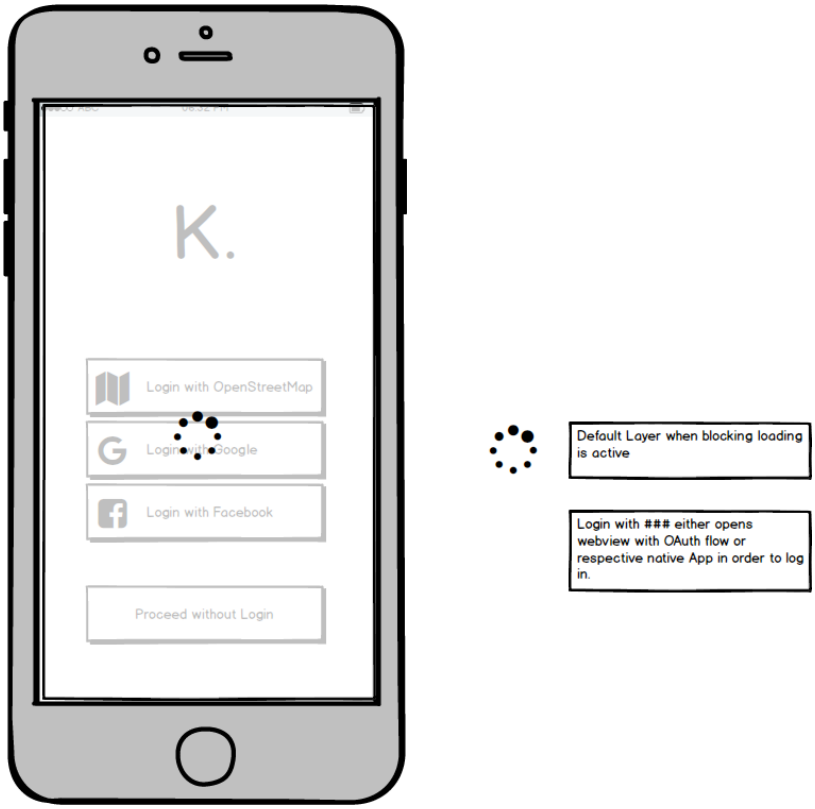

Figure 11.2.: Login Progress

Figure 11.3.: Logout

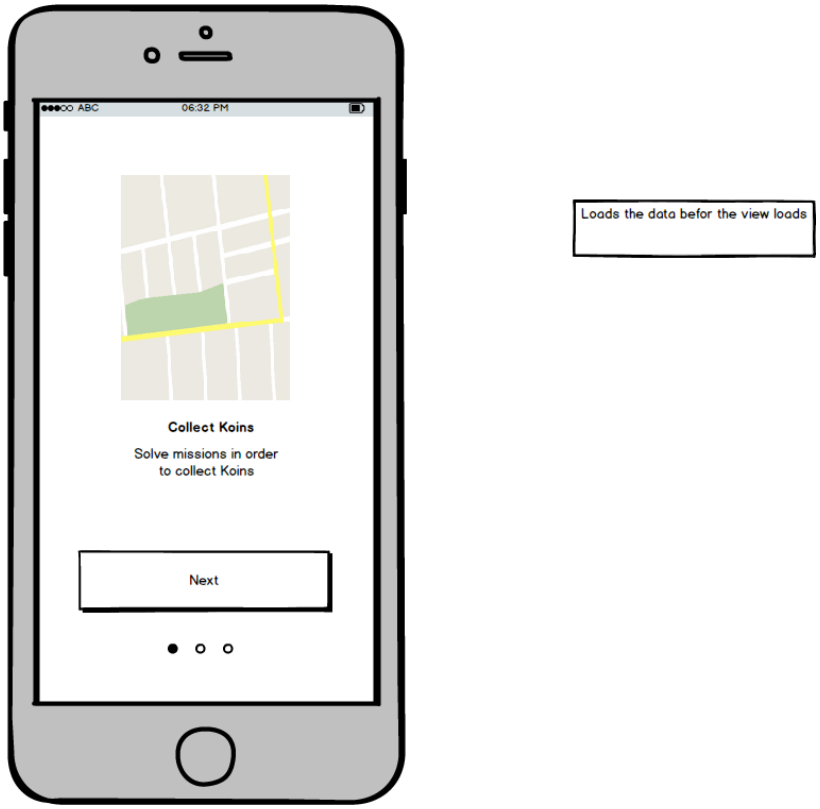

Figure 11.4.: Showcase

Figure 11.5.: Missions

Figure 11.6.: Missions

Figure 11.7.: Mission Numeric Answer

Figure 11.8.: Mission Text Answer

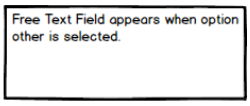

Figure 11.9.: Mission Text Combo

Figure 11.10.: Mission completed

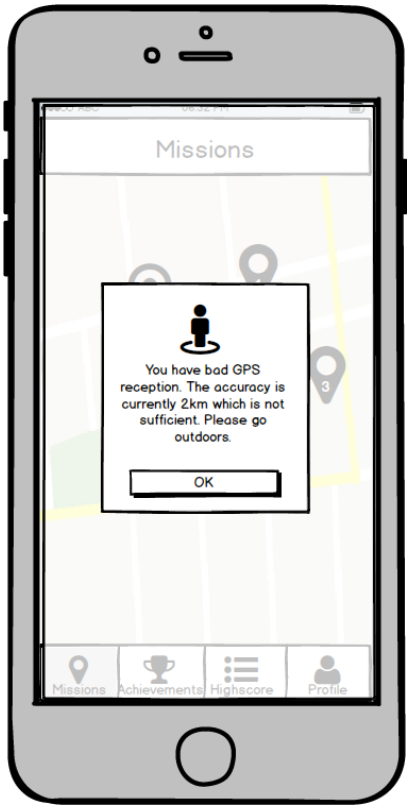

Figure 11.11.: Mission bad reception

Figure 11.12.: Solve Mission

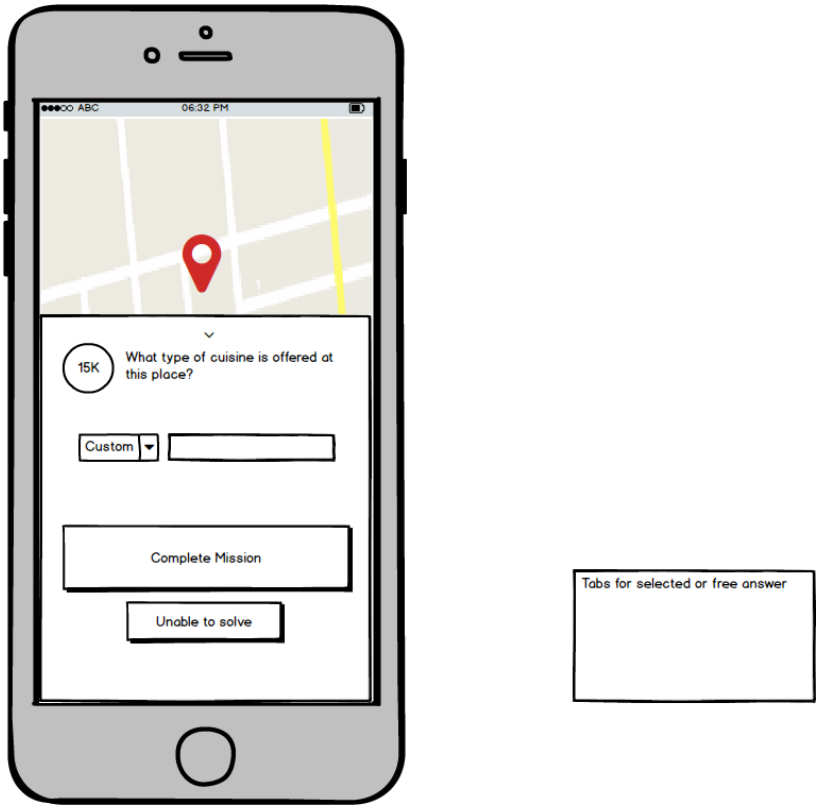

Figure 11.13.: Solve Mission

Figure 11.14.: Solve Mission

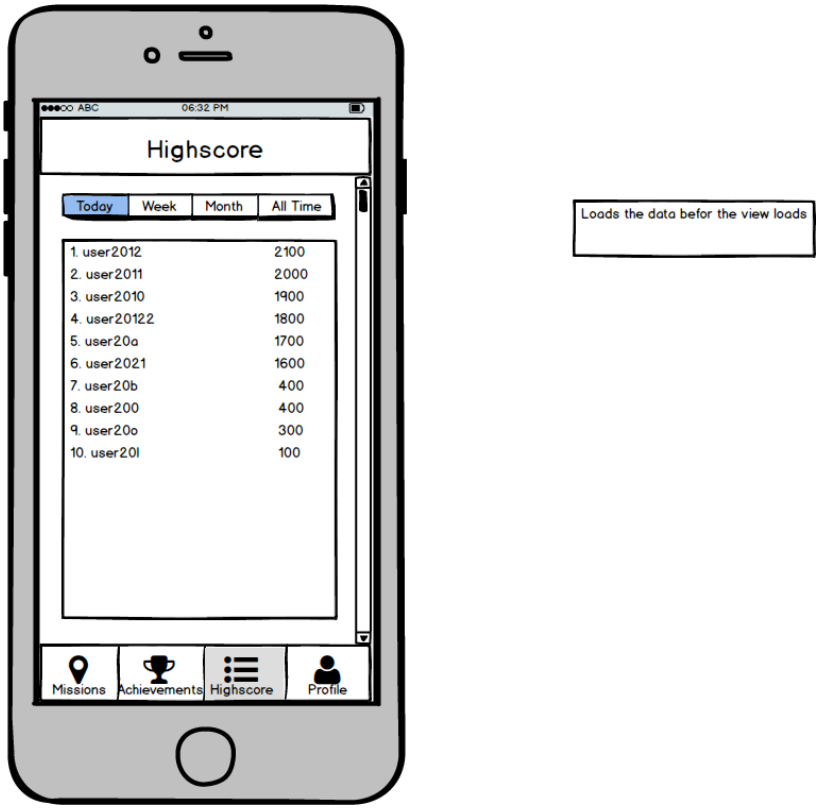

Figure 11.15.: Highscore

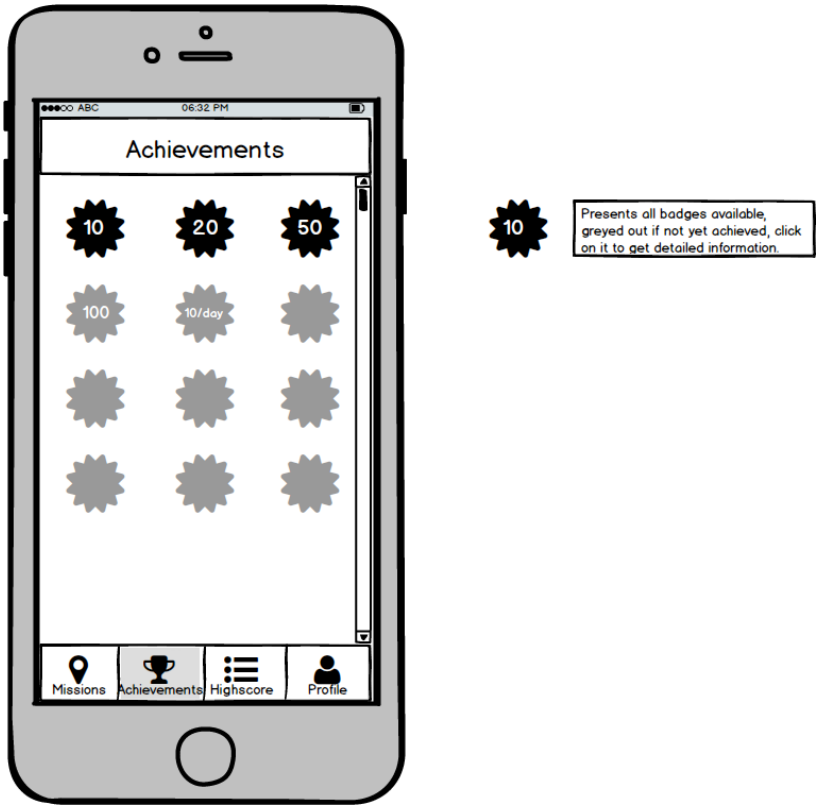

Figure 11.16.: Achievements

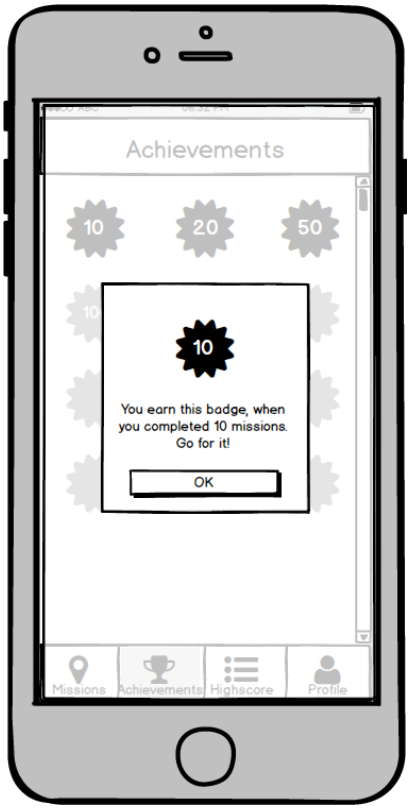

Figure 11.17.: Achievements

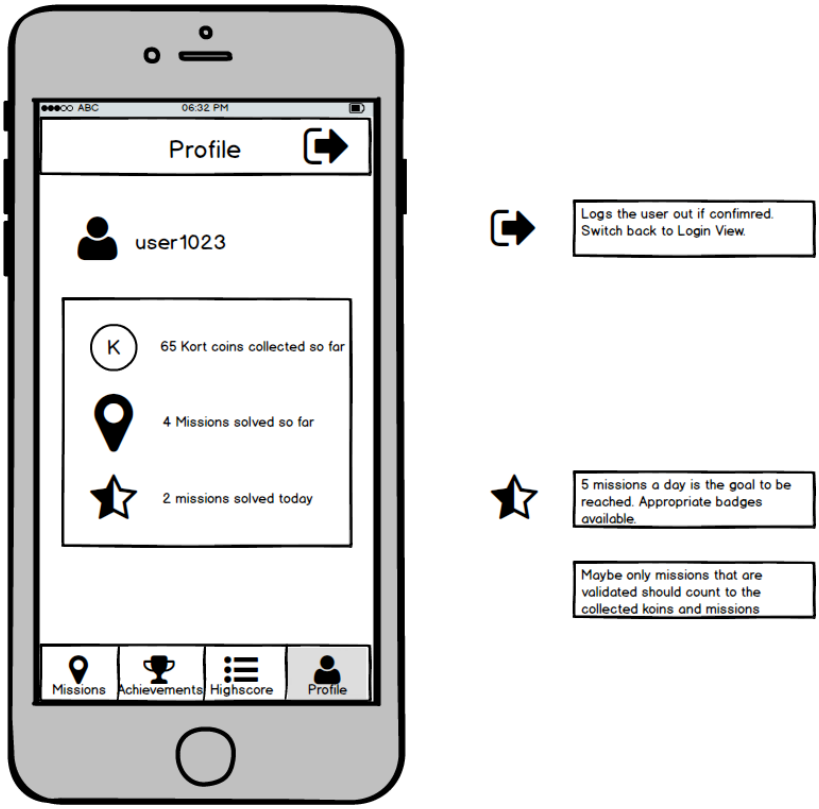

Figure 11.18.: Profile

Figure 11.19.: Settings

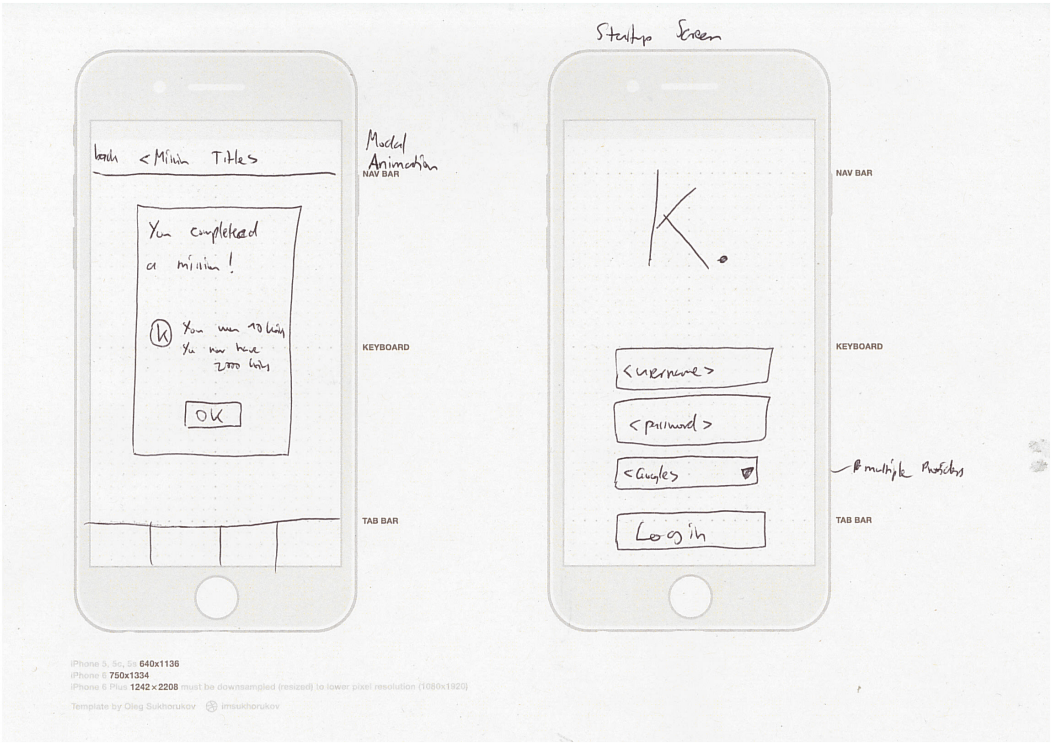

Figure 11.20.: Paper Prototype: Login & Mission Success

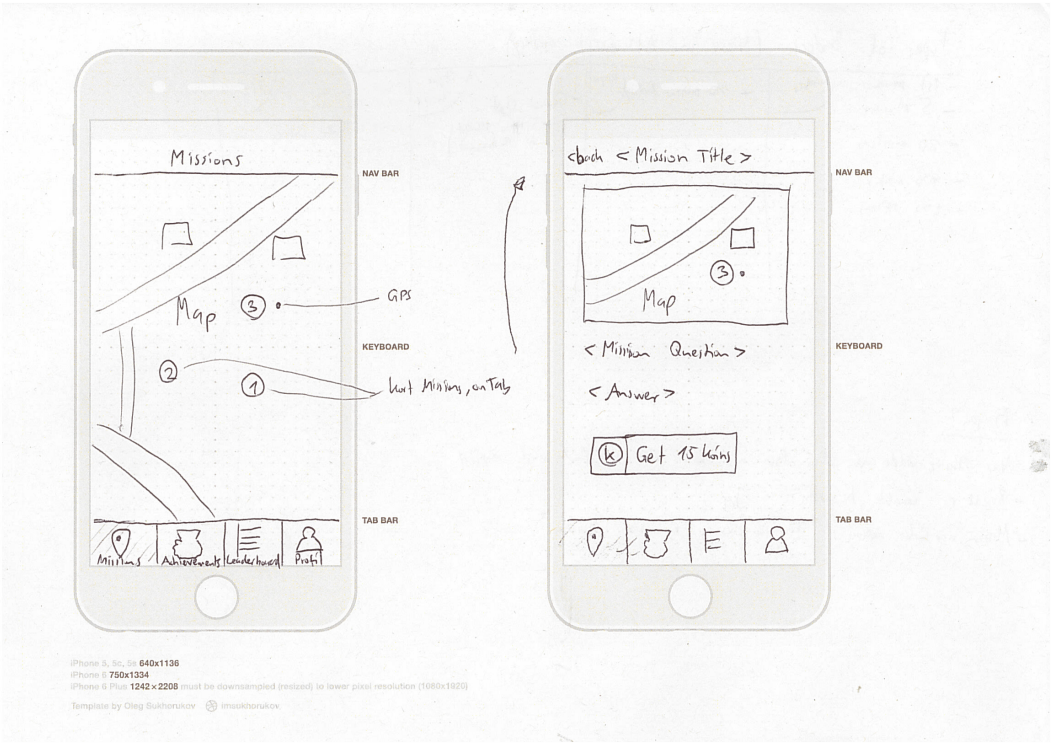

Figure 11.21.: Paper Prototype: Missions

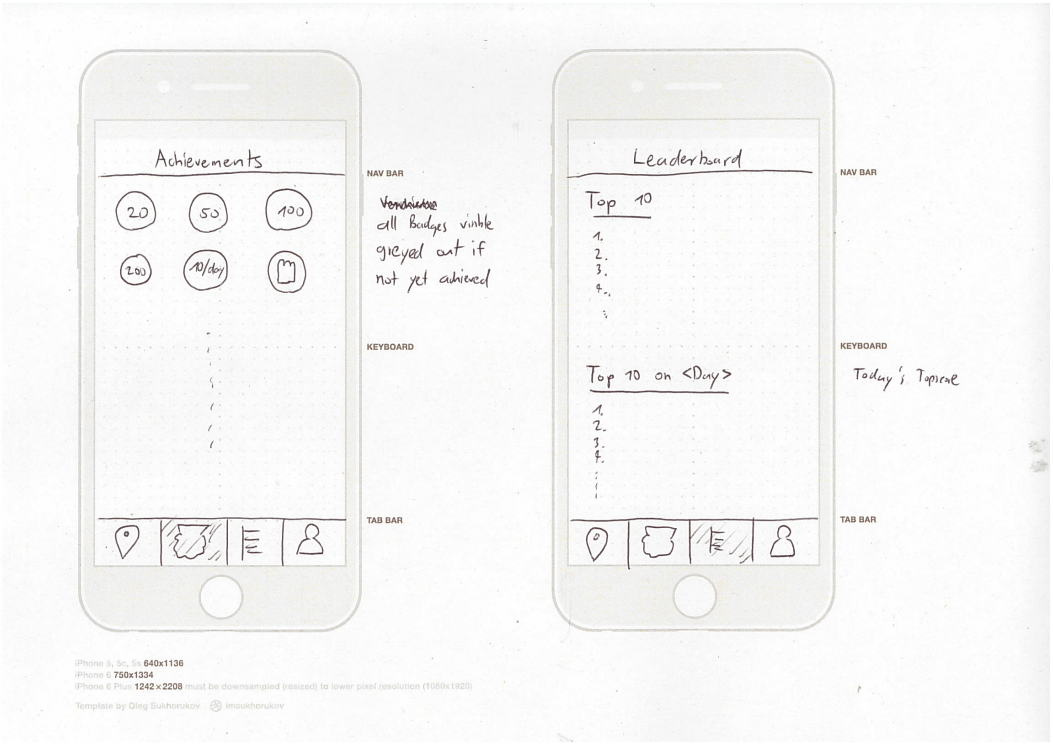

Figure 11.22.: Paper Prototype: Achievements & Leaderboards

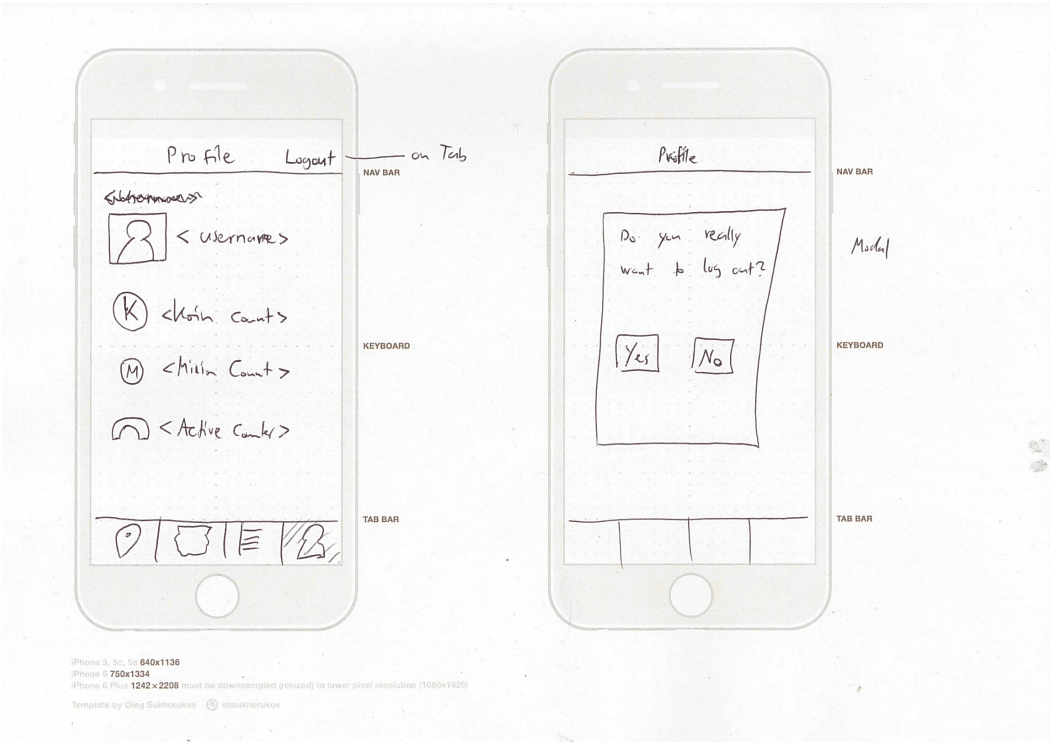

Figure 11.23.: Paper Prototype: Profile

12. Backend Documentation

The API specification generated by Swagger can be accessed here:

https://kort.dev.ifs.hsr.ch/v1.0/ui/#/

API calls can be directly formulated in this interface. For the sake of documentation, a static PDF export has been produced with the help of swagger2markup[1] which transforms the YAML specification file to AsciiDoc and asciidoctor-pdf[2] which transforms the AsciiDoc to PDF.

```
# get the binary
wget https://jcenter.bintray.com/io/github/swagger2markup/swagger2markup-cli/1.1.0/swagger2markup-cli-1.1.0.jar

# transform yaml to adoc
java -jar swagger2markup-cli-1.1.0.jar convert  -i kort-core/src/api/swagger.yaml --outputFile api

# transform adoc to pdf
asciidoctor-pdf api.adoc
```

Code Snippet 12.1: Swagger YAML to PDF conversion

[1] https://github.com/Swagger2Markup/swagger2markup
[2] https://github.com/asciidoctor/asciidoctor-pdf

Kort API

Overview

Version information

Version : 1.0

URI scheme

BasePath : /v1.0

Consumes

- `application/json`

Produces

- `application/json`

Paths

returns the achievements

```
GET /achievements
```

Parameters

Type	Name	Description	Schema	Default
Query	**lang** *optional*		string	`"en"`
Query	**user_id** *optional*	unique user id	string	`"-1"`

Responses

HTTP Code	Description	Schema
200	returns newly aquired achievements if any	< Achievement > array
400	bad request	No Content
500	internal server error	No Content

Tags

- Achievements

returns the current highscore

```
GET /highscore
```

Parameters

Type	Name	Schema	Default
Query	**limit** *optional*	integer	`"10"`
Query	**type** *optional*	enum (day, week, month, all)	`"all"`

Responses

HTTP Code	Description	Schema
200	returns the highscore	< Highscore > array
400	bad request	No Content
500	internal server error	No Content

Tags

- Highscore

returns all missions within the parameters given

```
GET /missions
```

Parameters

Type	Name	Description	Schema	Default
Query	**lang** *optional*		string	`"en"`
Query	**lat** *required*		number	
Query	**limit** *optional*		integer	`"100"`

Type	Name	Description	Schema	Default
Query	**lon** *required*		number	
Query	**radius** *required*		integer	
Query	**user_id** *optional*	unique user id	string	"-1"

Responses

HTTP Code	Description	Schema
200	Returns missions	< Mission > array
400	bad request	No Content
500	internal server error	No Content

Tags

- Mission

returns the geometry of the osm feature

```
GET /missions/osm/{osm_type}/{osm_id}
```

Parameters

Type	Name	Description	Schema
Path	**osm_id** *required*	the osm id	integer
Path	**osm_type** *required*	the osm type (node, way, relation)	enum (node, way, relation)

Responses

HTTP Code	Description	Schema
200	returns list of coordinates	No Content
400	bad request	No Content
500	internal server error	No Content

Tags

- Mission

submit solution to missions

```
POST /missions/{schema_id}/{error_id}/solution
```

Parameters

Type	Name	Description	Schema
Path	**error_id** *required*	error id	integer
Path	**schema_id** *required*	schema id	string
Body	**body** *required*		missionSolution

Responses

HTTP Code	Description	Schema
200	submit a mission solution, returns newly aquired achievements	< Achievement > array
400	bad request	No Content
404	an error occured	No Content
500	internal server error	No Content

Tags

- Mission

returns the statistics

```
GET /statistics
```

Responses

HTTP Code	Description	Schema
200	return the current statistics	Statistics
500	internal server error	No Content

Tags

- Statistics

get user information

```
GET /users/{user_id}
```

Parameters

Type	Name	Description	Schema
Header	**Authorization** *required*	the user secret given after succesful OAuth login	string
Path	**user_id** *required*	unique user id	integer

Responses

HTTP Code	Description	Schema
200	return user	User
400	bad request	No Content
401	unauthorized	No Content
404	user does not exist	No Content
500	internal server error	No Content

Tags

- User

Definitions

Achievement

Name	Description	Schema
achieved *optional*	whether or not the achievement has been achieved	boolean
achievementDate *optional*	the date on which the achievement has been achieved	string(date-time)
achievementDescription *optional*	achievement description	string
achievementId *optional*	achievement id	integer

Name	Description	Schema
achievementImageURI *optional*	uri for the badge icon	string
achievementTitle *optional*	achievement title	string

Highscore

Name	Description	Schema
id *optional*	id	integer
koin_count *optional*	how many koins this user has earned	integer
mission_count *optional*	how many missions this user has solved	integer
rank *optional*	the rank for this user	integer
user_id *optional*	the id of this user	string
username *optional*	the username	string

Mission

Name	Description	Schema
annotationCoordinate *optional*	lat/lon array of annotation	< number > array
error_type *optional*	mission type **Example** : `"missing_track_type"`	string
geomType *optional*	geometry type **Example** : `"line"`	enum (point, line)
id *optional* *read-only*	id consisting of the schema id and the error id **Example** : `"s95id59805085"`	string
image *optional*	the image file for the annotation either as URL or local filename in app **Example** : `"mission_road"`	string
inputType *optional*		inputType
koinReward *optional*	number of koins as reward **Example** : `5`	integer

Name	Description	Schema
koinRewardWhenComplete *optional*	number of koins when other answers are the same **Example** : `10`	integer
osmId *optional*	The unique OSM id	number
osmType *optional*		enum (node, way, relation)
question *optional*	question **Example** : `"What kind of track is this?"`	string
title *optional*	mission title for display **Example** : `"Type of track unknown"`	string

inputType

Name	Description	Schema
constraints *optional*		constraints
name *optional*	name of the input type **Example** : `"select"`	enum (number, select, text)
options *optional*	the selection to choose from	< string > array
values *optional*	the corresponding values of each selection	< string > array

constraints

Name	Description	Schema
description *optional*	the localized message that is shown to the user if validation fails **Example** : `"Number must be between 1 and 100"`	string
lowerBound *optional*	the lower bound if any **Example** : `1`	integer
re *optional*	the regular expression in place	string
upperBound *optional*	the upper bound if any **Example** : `100`	integer

Solution

Name	Description	Schema
koins *optional*	number of koins earned	integer

Name	Description	Schema
lang *optional*	**Default** : `"en"`	string
option *optional*	the option value if any	string
osm_id *optional*	the unique osm id	integer
solved *optional*	whether the mission was solved or not	boolean
userId *optional*	unique user id	integer
value *optional*	the answer value	string

Statistics

Name	Description	Schema
badge_count *optional*	the total number of achievements which have been obtained	integer
complete_fix_count *optional*	the number of solved missions which have been completed with the help of other answers and are ready to be submitted to OSM	integer
fb_user_count *optional*	self-explanatory	integer
fix_count *optional*	the number of solved missions (includes unsolved missions)	integer
fix_count_language_unknown_100_count *optional*	self-explanatory	integer
fix_count_language_unknown_50_count *optional*	self-explanatory	integer
fix_count_language_unknown_5_count *optional*	self-explanatory	integer
fix_count_missing_cuisine_100_count *optional*	self-explanatory	integer

Name	Description	Schema
fix_count_missing_cuisine_50_count *optional*	self-explanatory	integer
fix_count_missing_cuisine_5_count *optional*	self-explanatory	integer
fix_count_missing_level_100_count *optional*	self-explanatory	integer
fix_count_missing_level_50_count *optional*	self-explanatory	integer
fix_count_missing_level_5_count *optional*	self-explanatory	integer
fix_count_missing_maxspeed_100_count *optional*	self-explanatory	integer
fix_count_missing_maxspeed_50_count *optional*	self-explanatory	integer
fix_count_missing_maxspeed_5_count *optional*	self-explanatory	integer
fix_count_missing_track_type_100_count *optional*	self-explanatory	integer
fix_count_missing_track_type_50_count *optional*	self-explanatory	integer
fix_count_missing_track_type_5_count *optional*	self-explanatory	integer

Name	Description	Schema
fix_count_motorway_ref_100_count *optional*	self-explanatory	integer
fix_count_motorway_ref_50_count *optional*	self-explanatory	integer
fix_count_motorway_ref_5_count *optional*	self-explanatory	integer
fix_count_opening_hours_100_count *optional*	self-explanatory	integer
fix_count_opening_hours_50_count *optional*	self-explanatory	integer
fix_count_opening_hours_5_count *optional*	self-explanatory	integer
fix_count_poi_name_100_count *optional*	self-explanatory	integer
fix_count_poi_name_50_count *optional*	self-explanatory	integer
fix_count_poi_name_5_count *optional*	self-explanatory	integer
fix_count_religion_100_count *optional*	self-explanatory	integer
fix_count_religion_50_count *optional*	self-explanatory	integer
fix_count_religion_5_count *optional*	self-explanatory	integer

Name	Description	Schema
fix_count_way_wo_tags_100_count *optional*	self-explanatory	integer
fix_count_way_wo_tags_50_count *optional*	self-explanatory	integer
fix_count_way_wo_tags_5_count *optional*	self-explanatory	integer
google_user_count *optional*	self-explanatory	integer
highscore_place_1_count *optional*	self-explanatory	integer
highscore_place_2_count *optional*	self-explanatory	integer
highscore_place_3_count *optional*	self-explanatory	integer
id *optional*	id	integer
incomplete_fix_count *optional*	the number of missions which have not yet been completed with the help of other answers	integer
osm_user_count *optional*	self-explanatory	integer
six_per_day_count *optional*	self-explanatory	integer
solved_language_unknown_count *optional*	self-explanatory	integer
solved_missing_cuisine_count *optional*	self-explanatory	integer

Name	Description	Schema
solved_missing_level_count *optional*	self-explanatory	integer
solved_missing_maxspeed_count *optional*	self-explanatory	integer
solved_missing_track_type_count *optional*	self-explanatory	integer
solved_motorway_ref_count *optional*	self-explanatory	integer
solved_opening_hours_count *optional*	self-explanatory	integer
solved_poi_name_count *optional*	self-explanatory	integer
solved_religion_count *optional*	self-explanatory	integer
solved_way_wo_tags_count *optional*	self-explanatory	integer
total_fix_count_100_count *optional*	self-explanatory	integer
total_fix_count_10_count *optional*	self-explanatory	integer
total_fix_count_1_count *optional*	self-explanatory	integer
total_fix_count_50_count *optional*	self-explanatory	integer
user_count *optional*	self-explanatory	integer
validated_fix_count *optional*	the number of solved missions	integer

User

Name	Description	Schema
email *optional*	the user's email if any **Length** : 1 - 100 **Example** : "user@kort.ch"	string
id *optional* *read-only*	unique identifier **Example** : 12	integer
koin_count *required*	how many koins the user has obtained **Example** : 825	integer
last_login *required* *read-only*	Creation time **Example** : "2015-07-07T15:49:51.230+02:00"	string(date-time)
logged_in *required*	whether the user is logged in **Example** : true	boolean
mission_count *required*	how many mission the user has finished **Example** : 2	integer
mission_count_today *required*	how many mission the user has finished today **Example** : 2	integer
name *required*	user's full name **Length** : 1 - 100 **Example** : "user's name"	string
oauth_provider *required*	oauth provider **Example** : "google"	string
oauth_user_id *required*	oauth user id **Example** : "htksduwe23js"	string
pic_url *required*	url to avatar picture **Example** : "www.gravatar.com/avatar/secret?s=200&d=mm&r=r"	string
secret *required*	url to avatar picture **Example** : "asdojasdkjas"	string
token *required*	token from oauth **Example** : "FAF923jdsdDOaOs"	string
username *required*	the username used **Length** : 1 - 100 **Example** : "username"	string

missionSolution

Name	Schema
solution *optional*	Solution

Glossary

A/B

A/B testing is a controlled experiment with two variants, A and B. 41

API

Application Programming Interface. iv, v, 7, 10, 74, 75, 118

backend

The backend is an interface between the data and the frontend. It is also known as data access layer. iv, 3, 28, 74, 84, 119

crowdsourcing

Crowdsourcing allows the power of the crowd to accomplish tasks that were once the province of just a specialized few [20]. 3

CRUD

Acronym for create, read, update, and delete which are the four basic functions of persistent storage. 69

Deep Linking

With deep linking, an app can contextually redirect the user to a precise location in another app. vi, 33, 34, 77

DOM

DOM stands for Document Object Model. As an example, an HTML DOM consist of elements of HTML. It is an in-memory representation of rendered text. The HTML DOM then provides an API to traverse the tree and modify nodes [23]. 8, 10, 121

Framework

A framework usually includes a lot of libraries to make work easier. It dictates code and design methodologies. 3

frontend

The frontend is an interface between the user and the backend. It is also known as presentation layer. iv, 28, 29, 74, 118

gamification

The process of game-thinking and game mechanics to engage users and solve problems. Specifically, crowdsolving is a collaborative way of solving a problem using many people [17]. iii, vi, 3, 4, 77

Hybrid App

An approach to combine best from Web apps and Native Apps: Native code for hardware-specific and high-performance tasks and web technologies for the rest. 7, 10

I18n

I18n stands for Internationalization, stemming from I(18-letters)n. It means providing the environment for easily translating software projects. 27

IDE

IDE stands for Integrated Development Environment, a software application for the development of software. 7, 79

KISS

The KISS is an abbreviation of Keep It Stupid Simple or Keep It Simple, Stupid. A common principle in software engineering, stating that most systems work best if they are kept simple rather than made complicated [3]. 29, 67

linting

Linting is the process of running a program that will analyse code for potential errors. 83

microservices

Microservices is a variant of the glsSOA architectural style that structures an application as a collection of loosely coupled services [37]. 37

MMOG

Massively multiplayer online games are online games capable of supporting thousand of players simultaneously in the same virtual world. 47

Native App

Native apps are interactive and feature-rich, with full access to the hardware of the device. iii, vi, 7, 8, 77, 119

node

Nodes are points with a geographic position, stored as coordinates (pairs of a latitude and a longitude) according to WGS 84. Outside of their usage in ways, they are used to represent map features without a size, such as points of interest or mountain peaks [38]. 3, 120, 121

Node.js

Node.js is an open-source, cross-platform JavaScript run-time environment for executing JavaScript code server-side. 120

NPM

NPM is the default package manager for Node.js. 31

OAuth

OAuth is an open standard for access delegation, commonly used as a way for Internet users to grant websites or applications access to their information on other websites but without giving them the passwords [18]. v, 75

OSM

OpenStreetMap (OSM) is a collaborative project to create a free editable map of the world. The creation and growth of OSM has been motivated by restrictions on use or availability of map information across much of the world, and the advent of inexpensive portable satellite navigation devices [38]. iii–vi, 2, 74–76

relation

Relations are ordered lists of nodes, ways and relations (together called 'members'), where each member can optionally have a 'role' (a string). Relations are used for representing the relationship of existing nodes and ways. Examples include turn restrictions on roads, routes that span several existing ways (for instance, a long-distance motorway), and areas with holes [38]. 120, 121

REST

Representational state transfer (REST) or RESTful web services allow requesting systems to access and manipulate textual representations of web resources using a predefined set of stateless operations [16]. iv, 38, 74

software libraries

A software library is a suite of data and programming code that is used to develop software applications. 31

SPA

A Single-Page Application is a web application that fits on a single web page with the goal of providing a user experience similar to that of a desktop application. 67

tag

Tags are key-value pairs (both arbitrary strings). They are used to store metadata about the map objects (such as their type, their name and their physical properties) Tags are not free-standing, but are always attached to an object: to a node, a way or a relation [38]. 3

UI

UI stands for User Interface, specifying the design field for human-machine interaction. 7, 8, 10

Unit Test

Unit Testing is a software testing method with the goal of testing small parts of codes in so-called modules. It is tested whether the code is fit for use. 81, 82

Use Case

A Use Case is a list of actions or event steps, usually defining the interactions between an actor and a system to achieve a goal. 17

Virtual DOM

The Virtual DOM is an abstraction of the DOM, which is an abstraction itself. It is basically simplified and local copy of the DOM [23]. 8

way

Ways are ordered lists of nodes, representing a polyline, or possibly a polygon if they form a closed loop. They are used both for representing linear features such as streets and rivers, and areas(like forests, parks, parking areas and lakes) [38]. 3, 120, 121

Web app

A web application or web app is a client–server software application in which the client (or user interface) runs in a web browser. Common web applications include webmail,

online retail sales, online auctions, wikis, instant messaging services and many other functions [39]. iii, vi, 2, 3, 7, 76, 119

YAML

YAML stands for YAML Ain't Markup Language and is a data-oriented markup language. 35

Bibliography

[1] Docker Overview, 2017. [Online; Accessed 2 July 2017]. URL: https://docs.docker.com/engine/docker-overview/#what-can-i-use-docker-for.

[2] Full Stack Python - API Creation, 2017. [Online; Accessed 3 July 2017]. URL: https://www.fullstackpython.com/api-creation.html.

[3] KISS, 2017. [Online; Accessed 10 July 2017]. URL: http://people.apache.org/~fhanik/kiss.html.

[4] OpenAPI Specification, 2017. [Online; Accessed 29 June 2017]. URL: https://github.com/OAI/OpenAPI-Specification/blob/master/versions/2.0.md.

[5] React Native - Native Modules, 2017. [Online; Accessed 1 July 2017]. URL: https://facebook.github.io/react-native/docs/native-modules-ios.html.

[6] Ten minute introduction to MobX and React, 2017. [Online; Accessed 2 July 2017]. URL: https://mobx.js.org/getting-started.html.

[7] Wiktionary, The Free Dictionary, 2017. [Online; Accessed 29 June 2017]. URL: https://en.wiktionary.org/wiki/kort.

[8] Dan Abramov. Presentational and Container Components, 2017. [Online; Accessed 11 July 2017]. URL: https://medium.com/@dan_abramov/smart-and-dumb-components-7ca2f9a7c7d0.

[9] Luis Abreu. Why and How to Avoid Hamburger Menus, 2014. [Online; Accessed 13 July 2017]. URL: https://lmjabreu.com/post/why-and-how-to-avoid-hamburger-menus/.

[10] Richard Bartle. Hearts, clubs, diamonds, spades: Players who suit MUDs. Journal of MUD research, MUSE Ltd, Colchester, Essex, United Kingdom, 1996.

[11] Annrita Egli Carmelo Schumacher. Kort Reloaded. HSR Hochschule für Technik Rapperswil, 2013.

[12] Christopher Chedeau. Performance Calendar, 2013. [Online; Accessed 1 July 2017]. URL: https://calendar.perfplanet.com/2013/diff/.

[13] Sebastian Deutsch. Wie wählt man die beste Flux Implementierung, 2015. [Online; Accessed 2 July 2017]. URL: http://reactjs.de/posts/wie-wahlt-man-die-beste-flux-implementierung.

[14] Docker Docs. Best practices for writing Dockerfiles, 2017. [Online; Accessed 12 July 2017]. URL: https://docs.docker.com/engine/userguide/eng-image/dockerfile_best-practices/.

[15] Facebook. JSX In Depth, 2017. [Online; Accessed 1 July 2017]. URL: https://facebook.github.io/react/docs/jsx-in-depth.html.

[16] Roy Thomas Fielding. *REST: Architectural Styles and the Design of Network-based Software Architectures*. PhD thesis, University of California, Irvine, 2000. [Online; Accessed 29 June 2017]. URL: http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm.

[17] Gabe Zichermann, Christopher Cunningham. *Gamification by Design - Implementing Game Mechanics in Web and Mobile Apps*. O'Reilly Books, 2011.

[18] Whitson Gordon. Understanding OAuth: What Happens When You Log Into a Site with Google, Twitter, or Facebook, 2017. [Online; Accessed 29 June 2017]. URL: http://lifehacker.com/5918086/understanding-oauth-what-happens-when-you-log-into-a-site-with-google-twitter-or-facebook.

[19] Kevin Hoffman. *Beyond the Twelve-Factor App*. O'Reilly Books, 2016.

[20] Jeff Howe. *Crowdsourcing: Why the Power of the Crowd Is Driving the Future of Business*. Crown Business, 2008.

[21] Jesse Schell - Carnegie Mellon University. *The Art of Game Design: A Book of Lenses*. Morgan Kaufmann Publishers, 2008.

[22] Stefan Oderbolz Jürg Hunziker. Gamified Mobile App für die Verbesserung von OpenStreetMap. HSR Hochschule für Technik Rapperswil, 2012.

[23] Bartosz Krajka. React Kung Fu, 2017. [Online; Accessed 1 July 2017]. URL: http://reactkungfu.com/2015/10/the-difference-between-virtual-dom-and-dom/.

[24] Wei-Meng Lee. CODE Magazine, 2017. [Online; Accessed 1 July 2017]. URL: http://www.codemag.com/article/1401051.

[25] Andrzej Marczewski. Periodic Table of Gamification Elements, 2017. [Online; Accessed 23 July 2017]. URL: https://www.gamified.uk/2017/04/03/periodic-table-gamification-elements/.

[26] Dominic Mülhaupt Marino Melchiori. Kort Reloaded – A Gamified App for Collecting OpenStreetMap Data. HSR Hochschule für Technik Rapperswil, 2016.

[27] Dotan Nahum. *Programming React Native*. Leanpub, 2016.

[28] Tom Occhino. React Native: Bringing modern web techniques to mobile, 2015. [Online; Accessed 1 July 2017]. URL: https://code.facebook.com/posts/1014532261909640/react-native-bringing-modern-web-techniques-to-mobile/.

[29] UI Patterns. Navigation Tabs, 2009. [Online; Accessed 13 July 2017]. URL: http://ui-patterns.com/patterns/NavigationTabs.

[30] UI Patterns. Pull to refresh, 2009. [Online; Accessed 13 July 2017]. URL: http://ui-patterns.com/patterns/pull-to-refresh.

[31] Olaf Zimmermann (ABB Corporate Research). Making Architectural Knowledge Sustainable–Industrial Practice Report and Outlook. Invited IEEE Software Speaker, SEI SATURN, 2012. [Online; Accessed 9 July 2017]. URL: https://resources.sei.cmu.edu/library/asset-view.cfm?assetID=31345.

[32] Mike Riethmuller. Understanding flexbox, 2014. [Online; Accessed 1 July 2017]. URL: https://madebymike.com.au/writing/understanding-flexbox/.

[33] Robert Zubek Robin Hunicke, Marc LeBlanc. MDA: A Formal Approach to Game Design and Game Research. Nineteenth National Conference of Artificial Intelligence (San Jose, CA), 2004.

[34] Anthony Rose. UX designers: Side drawer navigation could be costing you half your user engagement, 2014. [Online; Accessed 13 July 2017]. URL: https://thenextweb.com/dd/2014/04/08/ux-designers-side-drawer-navigation-costing-half-user-engagement/#.tnw_43HwzVpk.

[35] The Modern Craft Studio. Crafting effective Microservices in Python, 2017. [Online; Accessed 13 July 2017]. URL: https://caricio.com/2016/09/16/crafting-effective-microservices-in-python/.

[36] Octo Technology. RESTful API Design, 2014. [Online; Accessed 13 July 2017]. URL: https://blog.octo.com/wp-content/uploads/2014/12/OCTO-Refcard_API_Design_EN_3.0.pdf.

[37] Wikipedia. Microservices, 2017. [Online; Accessed 13 July 2017]. URL: https://en.wikipedia.org/wiki/Microservices.

[38] Wikipedia. OpenStreetMap – Wikipedia, the free encyclopedia, 2017. [Online; Accessed 29 June 2017]. URL: https://en.wikipedia.org/wiki/OpenStreetMap.

[39] Wikipedia. Web application – Wikipedia, the free encyclopedia, 2017. [Online; Accessed 29 June 2017]. URL: https://en.wikipedia.org/wiki/Web_application.

[40] Zalando. RESTful API guidelines, 2017. [Online; Accessed 13 July 2017]. URL: https://zalando.github.io/restful-api-guidelines/index.html.

List of Figures

List of Tables

Listings