3D Route Builder

Bachelor Thesis

TU Delft

Computer Science and Engineering

Bachelor End Project

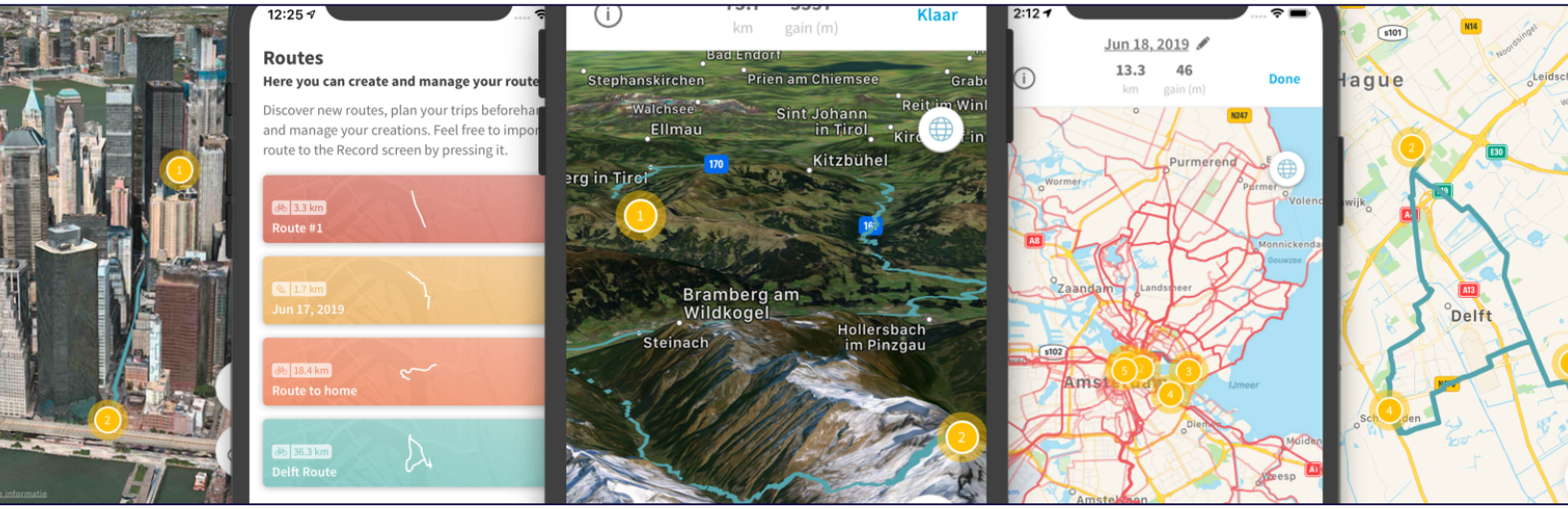
by

Rowdy Chotkan

Mika Kuijpers

Paul van der Laan

Brian Planje



Bachelor Project Coordinator

Otto Visser	TU Delft
Huijuan Wang	TU Delft

Bachelor Project Coach

Sander van den Oever	TU Delft

Client

Relive B.V.

Yousef El-Dardiry	Relive - Co-Founder
Frikkie Snyman	Relive - Project Supervisor

To be defended on July 2, 2019 at 4pm.

Abstract

Relive is a sports application that seeks to increase the retention rate of its user base. The project entails a route builder that can be incorporated in the Relive app. A route builder has several challenges, both in supporting user-friendly interaction, rendering the map and calculating the route. These challenges lead to design goals, such as a user experience focus, optimization of performance, and continuation of the project. To achieve these goals, the project is approached in an agile way with weekly planning meetings and tri-weekly stand-ups to keep it organized and well planned.

The front-end features are based on a specific target audience: the Relive users that do not use a third party app for sporting activities. The screen is intuitive and supports storing of planned routes. A route can be created, edited and loaded in the Record functionality of the Relive app.

To support the front-end, the back-end handles route calculation and route storing. The route calculation is done using Dijkstra's shortest path algorithm. In addition, a proof of concept customized route calculation is created that focuses on popularity oriented routing. Route storing makes use of the internal databases of Relive and is integrated in the existing systems.

Each feature is tested with automated end-to-end and unit tests. Additionally, user tests are performed to get valuable feedback from external users. Apart from active user participation, random users were selected to join a route builder experiment. During the project, more than 22 thousand users obtained the route builder feature. Each click is timed and tracked to make sure that the feature performed as expected.

Based on the events and user surveys, another iteration of the application was made by making improvements based on the acquired information. These improvements are validated using event tracking to measure the desired improvements.

Preface

This report describes the 3D Route Planner project concluded by Rowdy Chotkan, Mika Kuijpers, Brian Planje, and Paul van der Laan, at Relive. This project has been done as part of the Bachelor End Project for the Bachelor Computer Science and Engineering at Delft University of Technology. Over the course of 10 weeks, we have worked at Relive to build a fully mobile route planner for both iOS and Android, with 3D capabilities and a routing algorithm based on the most popular routes. This project has been a great learning experience for our software engineering, project planning, and time management skills. It has been quite an experience to put three years of our Computer Science study into practice in order to create a full software solution that actually aids people all over the world.

At Relive, we would like to thank Yousef El Dardiry and Lex Daniels for hosting this project and giving us the opportunity to take on this challenge. We also like to thank them for the great provided input, support, and advice. Furthermore, we would like to extend our gratitude to Frikkie Snyman for his excellent guidance, in planning, providing feedback, and his advice on implementation and design. Finally, we would like to thank all of Team Relive for their support, giving feedback, testing our application, and their overall hospitality.

At TU Delft, we would like to thank Sander van den Oever for his excellent guidance, feedback, and support throughout the project.

We hope that this report is able to guide the reader through the entire design process of the 3D Route Planner and get them as excited as we are.

Rowdy Chotkan,
Mika Kuijpers,
Brian Planje &
Paul van der Laan

Delft, July 2019

Contents

1 Introduction

Relive[1] is a relatively new sports application with more than 4 million users. With Relive, users can create a 3D movie of their sports activities. Even though Relive has a unique 3D movie feature, the competition is still strong in the app market (Pai & Li, 2014). Relive is currently focusing on improving their user retention rate, in order to build an advantage over the competition. Hence, the long term goal of this project is to increase the retention rate of the Relive app.
Considering the large user base, the new feature needs to be useful for a diverse amount of activities. Relive supports a limited number of activities but aims to increase this amount in the near future. Non-sporting activities can be considered for the new feature, for example road-tripping or vacations in general.

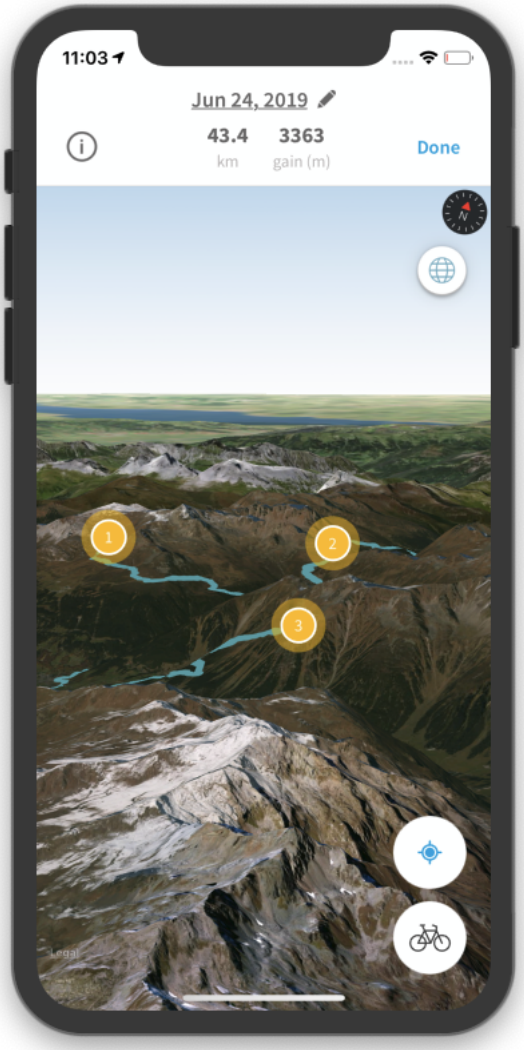


Figure 1: Route Planner Screenshot

The new feature is 3D route planning, such that users can easily plan new routes for their activities. The original project description can be found in Appendix J. Currently, users have to use third-party apps to plan their activities. The new feature allows users to plan their routes inside the Relive app and use the planned route immediately, whilst staying in the same app. The feature helps users to efficiently create new personalized routes, using the given tools and tips of the feature. A popular oriented routing algorithm, 3D maps, route statistics, user-friendly interaction, and more aim to provide the best route planning experience.

The new feature needs to function in the existing Relive app, which has implications for the available tools. The Relive app is mainly written in React Native[2], since it allows for a single code base that supports both iOS and Android. Relive has experience with 3D graphics, and therefore, has several suggestions for 3D libraries to use for the 3D route planning feature.

Additional libraries can be used to support the actual route planning. To achieve the best results for Relive app users, information such as route popularity can be taken into account. This requires customization or extension of the libraries that support route calculation.

In this report, research on the problem and possible solutions is defined in section 2. The organization and development processes are described in section 3. The design of the solution is described

[1]Website: https://www.relive.cc
[2]Website: https://facebook.github.io/react-native/

and justified in section 4. A general overview of the app is showcased in section 5 and the implementation of these features are elaborated upon in section 6. The testing methods and the results of this implementation can be found in section 7. To validate the requirements, several methods are used. These methods are described in section 8. Apart from internal validation, the SIG[3] provided valuable feedback, which is elaborated upon in section 9. The ethical implications of the solution can be found in section 10. Lastly, the discussion, conclusion and recommendations can be found in section 11, 12, and 13 respectively.

[3]Software Improvement Group

2 Research

This section includes the research and the corresponding solutions. First, the problem is defined and analyzed. After that, this section covers an analysis of the tools available to solve this problem. All options are analysed and the best fitting choices are explained.

2.1 Problem Definition

To build the route builder, several problems need to be solved. In this subsection, the problem is introduced. The problem statement was defined before the project started and refined in the first week. The problem is explored and a formal definition of the problem statement is stated.

2.1.1 Problem Statement

The project entails the planning of routes for sports trips such as hiking and cycling. Currently, Relive records routes during a sports activity, but includes no functionality to plan the trips beforehand. Users need to rely on third-party solutions to plan their routes. Relive also has a continuous goal of improving user retention rates for their application. This feature attempts to remove this third-party dependency and increase the user retention of the Relive app. As an additional unique and competitive edge, the map used for planning is visualized in 3D and fully available on mobile. This project will tackle the problem of route planning in a (3D) world map for different types of activities with a focus on cycling and hiking, with the additional long term goal of improving user retention rates.

This problem can be split up into three sub-problems:

1. Route calculation based on information such as activity type or road popularity.
2. User interaction with a map.
3. Map rendering.

By splitting up the problem, a more formal definition can be created. This formal definition will be discussed next.

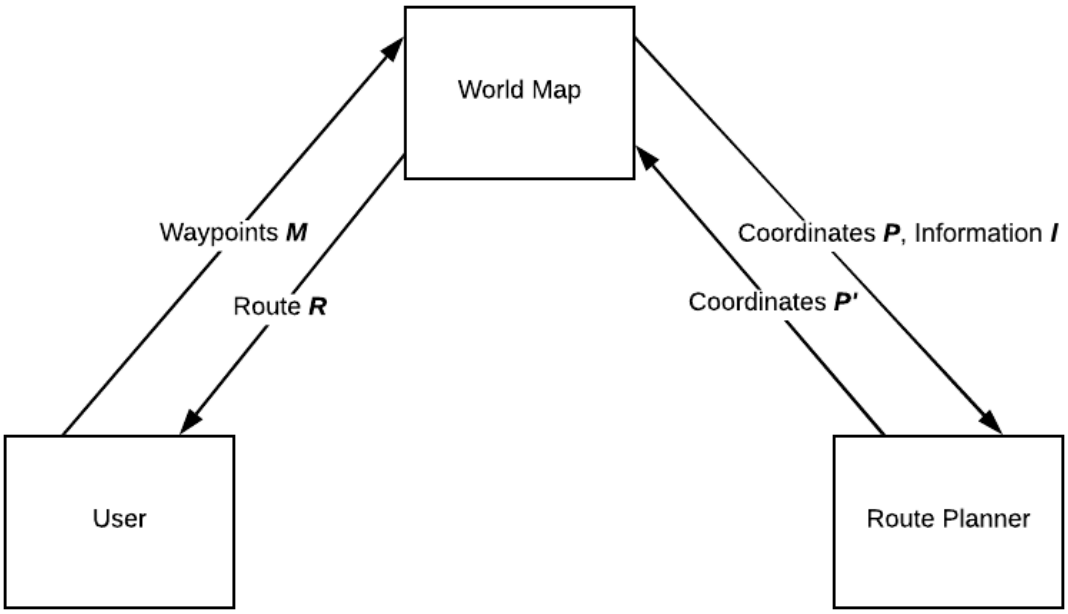


Figure 2: Problem Overview

2.1.2 Formal Definition

The aforementioned sub-problems (subsection 2.1.1) can be represented by the following actors:

- User: The agent interacting with the application.
- World map: The User Interface handling user input and visualization.
- Route Planner: The algorithm that calculates the routes.

The different actors coordinate as displayed in Figure 2. The User selects waypoints $M = m_1, m_2, ..., m_n$ with $n \geq 1$ on the World Map. Subsequently, the waypoints are converted into a set of coordinates P and are sent to the Route Planner. The Route Planner generates a valid route, represented by a set of coordinates P', based on the coordinates P. Planning a route can be based on minimizing the distance total distance of route R, given coordinates P. However, other route information (I) can be taken into account during the calculation, such as the popularity of certain routes. The coordinates P' are subsequently visualized on the World Map as Route R back to the User.

2.2 Problem Analysis

In this section, an analysis of the problem is given. Subsequently, a general description, the identified target audience, and a typical usage scenario are discussed.

2.2.1 Relive

Relive is an outdoor activity tracker application for both Android and iOS. It allows users to track their outdoor activities, such as running, cycling, hiking, skiing or snowboarding, and turn them into a video story. The video follows the route in a 3D landscape. Additional photos and statistics are shown in the video, such as Figure 3. The application is free, but also offers a subscription that enables additional features.

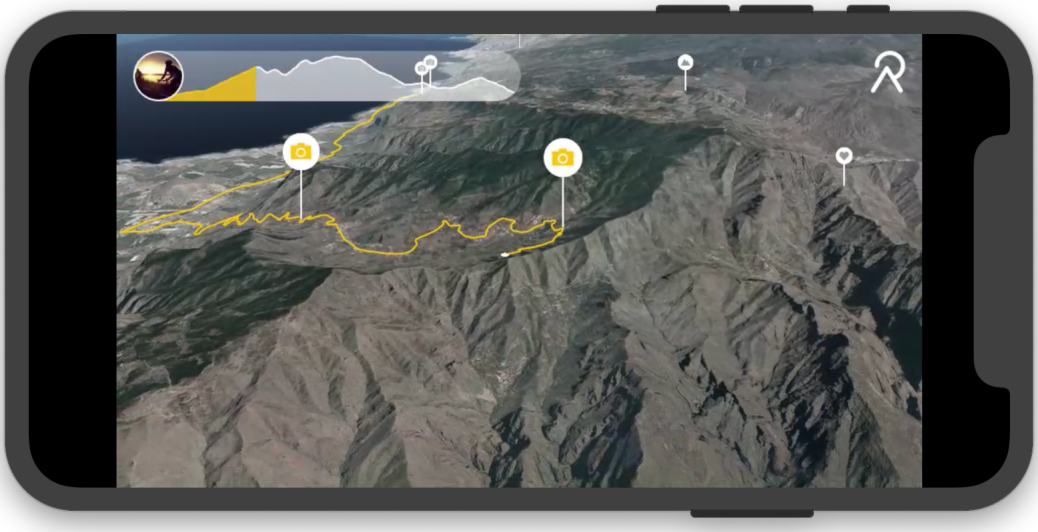


Figure 3: Relive 3D video

The existing code-base of Relive influences the approach and planning of the project. Relive's codebase is separated into two main repositories: the front- and back-end. The front-end deals with user interaction and displaying elements. It communicates with the back-end, which subsequently handles the data storage, processing, and analysis.

The users use Relive as a way to remember their trip and have a way of reliving this trip in the form of a video. Another reason is to share their trip with friends in a more appealing fashion, as opposed to simply having a map with a line, like a lot of the current tracking apps.

2.2.2 Project Environment

Together with the client, it was decided to develop the project inside the existing code base. This provides both benefits as well as drawbacks. The added benefits include a completely set-up development environment, allowing for quick prototyping and deployment to actual users, and easily requesting

code reviews from Relive developers. However, additional challenges include adhering to existing code styling rules and conventions, possible longer set-up time for getting the existing code base running and possible merge conflicts with the existing code base. Another important drawback is related to safety: in case the produced code contains vulnerabilities, it could have catastrophic effects for the Relive application if these flaws make it to production. Finally, Relive uses a weekly development cycle to release new features. This weekly cycle requires a corresponding feature development planning.

2.2.3 Target Audience

The largest user base of Relive participates in outdoor activities. This user base can be divided into two groups of people:

1. Performance-oriented users: people who perform a sport with the main goal of increasing their skill and/or performance over time.
2. Recreational users: people who perform a sport with the main goal of recreational use; focusing on the experience and not on the performance.

The audience group that is most focused on is group containing recreational users. According to Relive, there are more recreational than performance-oriented users and they are most in line with the functionality that the app provides, namely a focus on the experience instead of performance tracking. The possible target audiences for the route builder are slightly different:

1. Relive Record Users (uses the Record functionality of Relive)
   (a) External route planning app users
   (b) Users that only use the Relive app
2. External Record App users (merely uses the video render functionality of Relive)

The performance-oriented target audience of Relive roughly matches the External Record App users audience, since other apps provide more in-depth insights into their activities, which is useful for analyzing a performance-oriented sports activity. Recreational users fall in the Relive Record users category, which can be split up in two groups that are relevant for the route planner feature. The main focus is on Relive Record users that do not use an external route planning app. Based on experience relating to other Relive features, the retention rate of users can be increased by adding a new feature to the app.

2.2.4 Usage Scenario

The following usage scenario is an aggregation of the usage scenario's given in the interviews (Appendix H) and user stories (Appendix G).

Before a user goes out on a trip, the user has to decide on where to go. The user could decide where to go during or before the trip. It seems that a lot of users want to plan the route beforehand in order to see new places and to explore the environment. These users currently have to use a third party app to plan their route and use navigation during the trip to follow the route.

The Relive app or an external app can be used to record the trip of the user. Whenever the user decides to use the Relive Record feature, the user can see the currently traversed route on the map in the Record screen.

2.2.5 Custom route calculation

An important part of the route planner is the way in which paths are calculated between user-selected points on the map. The most obvious way is to calculate the shortest path between points, however, this may result in less optimal roads (e.g., non-paved roads when cycling).

Instead of calculating the shortest path, other information can be included in the route calculation to increase the quality of the planned route. The popularity of roads can be useful for users, as it filters out bad roads and prioritizes the well-known and proper roads. Relive has a considerable database that contains over [redacted] routes of Record users. These routes can be used to derive the popularity of roads. Other information might be considered as well, for example the road types.

The main trade-off is that popularity prioritization can increase the distance between the points that the user picks. Therefore, the user has less control over their route. In minimizing this consequence, it is important to analyze various algorithms that include popularity in route calculation by distance and popularity.

An important part of custom route calculation is experimenting with options and analyzing the results. Each user can prioritize certain factors, while those factors might be redundant for other users. A dynamic approach to support a wide range of users can be helpful to increase the usefulness of the app.

2.2.6 Map Matching

As mentioned in the previous section, the routes of the activities of the Relive users will be used in a custom routing algorithm. These routes are recorded through the built-in Relive tracker. The problem is that the custom routing algorithm uses OpenStreetMap (OSM) data for route calculations and GPS locations do not match OSM locations. In order to obtain the correct OSM locations, the recorded locations need to be matched. This problem is known as the map matching problem[4].

[4]Map matching problem: https://en.wikipedia.org/wiki/Map_matching

2.3 Development Tools

In this section, development tools will be discussed. First, the frameworks used to develop and deploy the application will be discussed. Afterward, the tools used to create the application will be discussed. There are two main types of tools that are used during development: mapping and routing tools. Mapping tools are the tools or frameworks used to visualize the map for the user, whilst routing tools are the tools or API's that are used to create the routes for the user.

2.3.1 Mobile Development Technologies

In this section, it will be discussed what technologies are available for mobile development and which of them is the most optimal for the solution, which, given the requirements, must be an Android and iOS application.

- **Android SDK (Java)**[5]
  The Android SDK uses Java and is used to develop Android applications.
  The main benefits are:
  - Prior experience with Java by all team members, which will bring the project up to speed quickly.
  - Ability to provide a native experience for Android users resulting in a faster and smoother user experience, which works towards the performance design goal.

  The main drawbacks are:
  - No support for iOS users.
- **iOS SDK (Swift/Objective-C)**[6]
  Native development for iOS applications is done using the iOS SDK, generally done with the usage of the programming language Swift.
  The main benefits are:
  - Ability to provide a native experience for iOS users.

  The main drawbacks are:
  - The members of the team don't have much experience with Swift or Objective-C programming.
  - In order to develop an application for iOS using Swift or Objective-C an Apple device is needed.
  - The iOS SDK can only be used for iOS and not for Android.
- **Progressive Web Applications (Web)**[7]
  Progressive Web Applications are an upcoming technology able to create native-like experiences for mobile exclusively through the use of web technology (HTML/CSS/Javascript). This is done by enabling a number of native features on mobile such as notifications and offline usage.
  The main benefits are:

[5]Website: `https://developer.android.com/studio`
[6]Website: `https://developer.apple.com/ios/`
[7]Website: `https://developers.google.com/web/progressive-web-apps/`

- Prior experience with web technologies by all team members which will bring the project up to speed quickly.

The main drawbacks are that:

- The user experience lacks compared to a native application since it essentially still is a web-page and is unable to make use of native components of the device. A web page can not make use of all the system features of a phone, compared to a native application. With the user experience design goal in mind, this is a less user friendly experience.

- **React Native (Javascript)**[8]
  React Native is a framework that allows building native mobile apps using JavaScript and React[9]. With React being a JavaScript library used to build interactive user interfaces, it is commonly used for websites. In contrast to Progressive Web Applications, React Native renders native controls which makes the experience comparable to native implementations such as the Android SDK or the iOS SDK.
  The main benefits are:
  - Cross-platform development for iOS and Android. This greatly increases the development power since the alternative is creating an application for iOS and Android separately.
  - Prior experience with Javascript/Typescript in the team will allow for a faster ramp-up time during development.
  - Easier interoperability and possible integration between the existing application and our solution, since the original application is made with this technology.
  - Native user experience since React Native uses the native components of the respective mobile device (iOS/Android).

  The main drawbacks are:
  - The amount of available libraries for React Native components is not as big as for example the Android SDK or the web, due to the smaller size of the development community.
  - Android and iOS respectively have some functionality that the other platform does not have. When using these functionalities, platform-specific code will have to be written.

From this selection of technologies React Native is the most optimal, due to the fact that it aligns best with the design goals we have formulated in subsection 4.2 and the fact that we have prior experience with the used language Typescript. This will most likely result in a faster development time, which is useful to combat the small time frame of 10 weeks.

2.3.2 Map Visualization Tools

There exist several map solutions that can be used in this project. In Table 1 an overview of alternatives is provided. As can be seen in the table, features do not differ much between native tools and third-party tools, therefore user experience was considered a higher priority in the final decision. There has been decided to use native maps instead of a third-party map visualization tool since these will most likely result in the best user experience. For iOS devices, Apple Maps will be used, which has 3D maps and for Android devices, Google Maps is the chosen alternative.

[8]Website: `https://facebook.github.io/react-native/`
[9]Website: `https://reactjs.org/`

	Google Maps	Apple Maps	Leaflet
3D	No	Yes	No
Mobile friendly	Yes	Yes	Yes
Free	Yes	Yes	Yes
Offline	Yes	Yes	Yes
iOS	Yes	Yes	Yes
Android	Yes	No	Yes

Table 1: Map Visualization Tool Comparison

2.3.3 Routing Tools

As mentioned in the problem statement, a sub-problem that needs to be solved is the route creation for different types of activities given a set of coordinates and a map. There are multiple options available which will discussed briefly, followed by a comparison and reasoning for the final choice.

	Google Directions	Apple MapKit	GraphHopper	Valhalla	OSRM
Map	Google Maps	Apple Maps	OSM[10]	OSM	OSM
Running/Walking	Yes	Yes	Yes	Yes	Yes
Bicycling	Yes	No	Yes	Yes	Yes
Driving	Yes	Yes	Yes	Yes	Yes
Number of activities	4	3	9	13+	3+
Waypoint support	Yes	No	Yes	Yes	Yes
Directions	Yes	Yes	Yes	Yes	Yes
Multiple route suggestions	Yes	Yes	Yes	Yes	Yes
Offline routing	No	Yes	No	Yes	Yes
Round trips with 1 point	No	No	Yes	No	No
Custom route suggestions	No	No	Yes	No	No
Cost	€0,009/request	Free*	€304/month	Free	Free

Table 2: Routing tools options

** The MapKit API is free but in order to publish an application, one requires a paid developer account. However, the company already owns such an account and thus Apple Mapkit can be considered free to use.*

Google Maps

Google's tool is called Google Maps which includes Routing support through a Directions API (*Developer Guide | Directions API*, n.d.). The tool supports driving, walking, cycling, and public transport. An interesting feature of the Directions API is related to winter sport. According to TechCrunch (*Google Adds Trail Maps For 100 Additional Ski Resorts To Google Maps*, n.d.), Google added ski routes, which indicates that winter sport routes are a possibility. The Google documentation (*Developer Guide | Directions API*, n.d.) supports gondola lifts (which are used in ski areas)

as a vehicle type as well. Currently, the downside is that Google does not support a wide range of sporting activities. However, it possibly could be added in the future and most activities fall under the general activities that Google supports.

Apple MapKit
Apple's tool is called MapKit and has a wide range of features that developers can use (*MapKit | Apple Developer Documentation*, n.d.). The supported transport types are limited to driving, walking, public transport, and 'other'. The major drawback is that Apple does not support cycling, which is a common activity type for Relive app users. Similarly to Google's tool, a route is calculated between two points, multiple route suggestions can be given, and directions are provided to follow the route.

GraphHopper
GraphHopper is a routing tool that uses OpenStreetMap as map source (*GraphHopper Directions API with Route Optimization*, n.d.; *OpenStreetMap*, n.d.). Most features are similar to the previous tools. Additionally, GraphHopper supports a wide range of activity types: car, truck, scooter, foot, hike, bike, mountainbike, motorcycle, and racingbike. GraphHopper has custom algorithms that are useful for a route planner. It supports round trips (same origin and destination) and alternative routes (all different fastest routes). Additionally, specific distances, random seed (for multiple route generation) and ranges (how much longer the alternative routes is allowed to be) can be specified to customize and personalize the route.

OSRM
OSRM (Open Source Routing Machine) (*Project OSRM*, n.d.) is an open source routing engine implemented using C++. The main focus of this project is the free-to-use nature as shown by the open source code and support for OpenStreetMap. The engine needs to be self-hosted which will add complexity compared to a service, however, this does allow a great degree of customization. The built-in transport types supported are driving, cycling, and walking. However, through the use of custom made profiles, new types of transport can be supported.

Valhalla
Vahalla (*Valhalla Routing Engine*, 2019) is also an open source routing engine implemented in C++. While it does use a different routing algorithm compared to OSRM, it also offers many of the same features such as multiple waypoints and directions. The main difference, compared to OSRM, is that Valhalla contains more built-in features, such as the largest variety of built-in activity types and narrative based turn-by-turn navigation in multiple languages.

The tools Google Directions and Apple MapKit will not be used since the number of activities is very limited compared to the other alternatives and since they are more focused on transport instead of sport and recreational activities. GraphHopper, Valhalla, and OSRM are all strong contenders in terms of activity types. After experimenting, it was decided to use GraphHopper since GraphHopper has the best support for activities, round trips, and allows for custom routing algorithms. Round trip support provides a valuable opportunity, as it allows a route to be calculated given one point and a distance. This can act as a route suggestion tool.

2.4 Map Matching Tool

The previously chosen routing engine, GraphHopper, also provides a library that implements an algorithm that handles map matching [11]. The algorithm that is implemented in this library is explained by (Newson & Krumm, 2009) and is described as follows:

> *This paper describes a novel, principled map matching algorithm that uses a Hidden Markov Model (HMM) to find the most likely road route represented by a time-stamped sequence of latitude/longitude pairs. (Newson & Krumm, 2009, p. 1)*

Two important requirements to ensure a high accuracy in the algorithm are that the sample interval and the noise in the tracker routes are minimized. The current tracker implementation works only with a GPS location service. It does not use multilateral location tracking from cell phone towers which would be too inaccurate for the algorithm. Furthermore, according to Relive the sampling rate is approximately 1 second which means that every second a location is recorded. This would amount to an error rate that is close to 0% (Newson & Krumm, 2009, p. 7), which is accurate enough for our use-case.

2.5 Web Server Technologies

In this section the research and the choice on the web framework will be elaborated upon. There are hundreds of frameworks available, thus a number of conditions are set-up to ensure a good fit for the project. The first condition is that it must be a major framework in order to ensure that there exists enough community support for issues that will eventually arise. There are a number of major frameworks to choose from as can be seen in the latest Stackoverflow Developer Survey[12]. This leads to a number of options in the top 10: Express, Spring, and Django in particular. Respectively these frameworks use Javascript, Java and Python which are all mastered by the members in the team. The frameworks themselves are comparable functionality wise, since they are all written in a major programming language with a large amount of libraries available. From this selection Express was chosen, due to the fact that it allows the team to write in the same programming language on both the front- and back-end, namely TypeScript. This enables higher productivity since any team-member will be able to write in either code bases without the need to switch language contexts.

2.6 Overview

To conclude, an overview is given in Table 3 on the major tools chosen to be used based on the research in this section,

	Programming Language	**Library / Runtime**
Front-end	Typescript	React Native (Library)
Back-end (web-server)	Typescript	Node JS (Runtime)
Back-end (routing engine)	Java	GraphHopper (Library)

Table 3: Overview of the major technologies that will be used in the project

[11] GraphHopper Map Matching Library: https://github.com/graphhopper/map-matching
[12] https://insights.stackoverflow.com/survey/2019#technology

3 Project Organization Processes

During the software development, organizational processes are essential to deliver the correct product on time and to handle the changing technical circumstances and demands of the client. Relive has a certain organizational approach towards new projects, which was used as inspiration for the project organization together with knowledge of Bachelor courses on Software Engineering. In this section, we cover the overall project organization processes.

3.1 Agile Development

Throughout the project, a form of Agile development is used to structure the process and increase value. According to (Highsmith & Cockburn, 2001), Agile developments shows slightly better business, quality, and customer performance and increases morale. This is based on a survey result of 200 people from a wide range of organizations. Additionally, we have learned that Agile development increases the ability to react to requirement changes during our study Computer Science and Engineering. The last point is particularly relevant for the route builder project, as the initial requirements are created at the start of the project without user input. In the last part of the project, user input is collected and has requirements changes and additions as a consequence.

Since the project allows for daily face-to-face communication, a less common approach to SCRUM is used. The word SCRUM is not specifically mentioned during the project, but the used approach and SCRUM show similarities. (Schwaber, 2004, p. 5-8) argues that the heart of Scrum is iteration. The project involved weekly iterations of structured planning. This was done in the form of tri-weekly progress meetings. During the week, daily meetings took place in an informal face-to-face fashion. The team and supervision figured out the tasks for the week based on the backlog and the team was free to choose how to achieve those tasks during the iteration. The backlog was kept in the form of MoSCoW product requirements, as visible in subsection 4.1, and was updated for all requirement changes.

Although the Agile development approach sounds like traditional SCRUM, the daily meetings were informal and unstructured. Additionally, as a traditional sprint is 30 days according to Ken Schwaber, the route builder project had informal sprints of 1 week. This approach was possible due to the small team and face-to-face communication possibilities. Due to these advantages, all uncertainties were easily solved and known by the whole team. The informalities allowed for faster iterations and less planning overhead. As mentioned before, SCRUM was not specifically mentioned during the project. The project can be seen as an informal approach to SCRUM based on the similarities and differences.

3.2 Meetings

A number of meetings have been held with the team members of Relive on various topics which all had an influence on our decision making. The most important meetings will be discussed here.

As discussed in subsection 3.1, tri-weekly progress meetings were held in the Agile development process. Each Monday, items from the backlog were confined and defined into tasks for that week. Each task had an assignee and detailed description if needed. The MoSCoW requirements functioned

as backlog. The organization of these tasks was performed with the help of the tool Asana[13]. This is a project management tool used within Relive and allows for delegation and organization of tasks. The logbook of the tasks during the project can be found in Appendix L.

In addition to the previously mentioned formal meetings, there are three informal stand-ups at Relive per week we participated in. In these meetings, everyone updates the team on what their weekly goals are and the status of these goals. The knowledge of the status of the team members helps in gauging whom to collaborate with or ask help from that week.

Remarkable meetings

During the project, some meeting had an impact on the development process. These meetings present an important part of the project organization process.

The project commenced with a roadmap meeting, in which the requirements of the client were discussed. Out of these requirements came the aforementioned (in subsection 3.1 and subsection 3.2) MoSCoW model. The requirements were continuously discussed with the client in order to fine-tune and improve the prioritization of the MoSCow. Another roadmap meeting was held in Week 5 of the project, in which the priority slightly shifted and features were re-prioritized as necessary. This is discussed more thoroughly in subsection 4.4.1.

The product manager meeting, having occurred in the middle of the project at the 14th of May, with Lee, discussed whether the requirements that were set-up were still on track on solving the real problems of the users. Other aspects that have been discussed are ways of validating whether demands are met using experiments, interviews, and surveys which are further elaborated upon in section 8.

In order to improve the flow and user-experience within the application, a meeting with the designer of Relive, Sander, was held on the 7th of June. During this meeting, the flow of the application was discussed and each User Interface (UI) element was introduced. Out of this meeting came several points of improvements on User Experience (UX) and UI. These results will be discussed in subsection 8.5.

In week 6, after about half of the project was concluded, a midterm-review was held. This review accompanied the client and supervisor from Relive and the coach from the TU Delft. During this meeting the progress of the project was shown, future plans were discussed and feedback was given.

[13]Website: `https://asana.com/`

4 Design

In this section, the initial design of the system and its requirements are stated and explained. This design is derived from the research done in the earlier parts of the project. The section will start with a set of initial requirements. For clarity, the design goals of the front- and back-end are discussed separately. During the project, some requirements and focuses shifted to other elements. These changes are documented in the last part of this section.

4.1 Requirements

A set of requirements is created for the project which lays out which features are essential and which are not. In order to prioritize tasks and build a road-map of all the to be implemented features, the 'MoSCoW' methodology was used[14]. This is created based on the user stories (Appendix G), the problem statement, and the requests from the client. This subsection contains the requirements and the reasoning behind the major requirement choices. Furthermore, a summary of the initial requirements is given. As expected, the requirements shifted during the project, which is discussed in subsection 4.4.1. A more detailed overview of the requirements can be found in Appendix C.

Relive follows the Minimal Viable Product (MVP) methodology. According to Relive and (Moogk, 2012), an MVP is a way to get a product as fast as possible on the market for user tests, which allows for user feedback. This way, requirements can be changed accordingly early on in the development process.

The MoSCoW and MVP work together, as the 'Must Haves' of the MoSCoW are equal to the MVP. The MVP of the route builder consists of an interactable 2D map that allows the user to plan and view a route. That route must be visible in the Record functionality of the app, so it can actually be used and followed during an activity. Additionally, the application must be able to calculate a route between the given waypoints.

Apart from the MVP, the MoSCoW contains more tasks that extend the functionality of the route builder. For example: the routes should be stored in a personal route list that a user can access and edit; additional information of the route should be visible; and the map should support 3D functionality.

Low on the priority are the Could Haves in the MoSCoW. These tasks include a heatmap of other routes, customized routing calculation, more route information, collaborative functionality, and extended personal route list organization functionality.

The 'Would Not Haves' of the MoSCoW consist of interesting ideas that can be useful, but require too much work for the duration of the project. These include navigation of the route, transforming a planned route into a 3D Relive video, and alternative route suggestions.

[14]MoSCoW https://en.wikipedia.org/wiki/MoSCoW_method

4.2 Design Goals

Three design goals are defined which are adhered to during the development process. During the research phase, these goals aid in defining the requirements by prioritizing certain options over others.

4.2.1 User Experience

A major design goal that has been identified is optimizing the user experience in such a way that the user can intuitively use the solution without a learning curve. A method that is focused on to improve this is continuous in-house testing of the user experience and using the results to incrementally improve the product.

During the analysis of the current Relive application, it became apparent that the app itself strictly enforces this design goal as well. This can be derived from the fact that the application, in essence, has one main feature users come for: delivering 3D videos from a fitness trip. This allows the application to have a small set of controls. Since the goal is that the solution will eventually be integrated into the Relive application, it is important to adhere to this design goal when developing our solution.

4.2.2 Performance

The solution that is to be delivered should be as lightweight and fast as possible. The reason for this is that, as stated, the solution will be implemented on Android and iOS mobile devices, which, especially Android devices, have a wide range of hardware capabilities. The final solution will need to have good performance on this range of devices in order to deliver a good user experience. This entails elements such as non-excessive load times, smooth animations and smooth visualizations of the map when interacting.

4.2.3 Continuation

The continuation of the project entails that the solution, preferably, is able to be integrated into the original application. This means that there has to be focused on non-functional requirements which will aid the integration. This includes testing, code interoperability with the original code-base, and code style adherence. Functionality wise, the overall engineering style of the solution must be as similar as possible to the original application.

4.3 High Level Overview

The high level design of the project consists of a front- and back-end (Figure 4). The front-end will be a React Native application, as discussed in subsection 2.3, which will handle the user interface for the mobile applications. The back-end will consist of three separate parts:

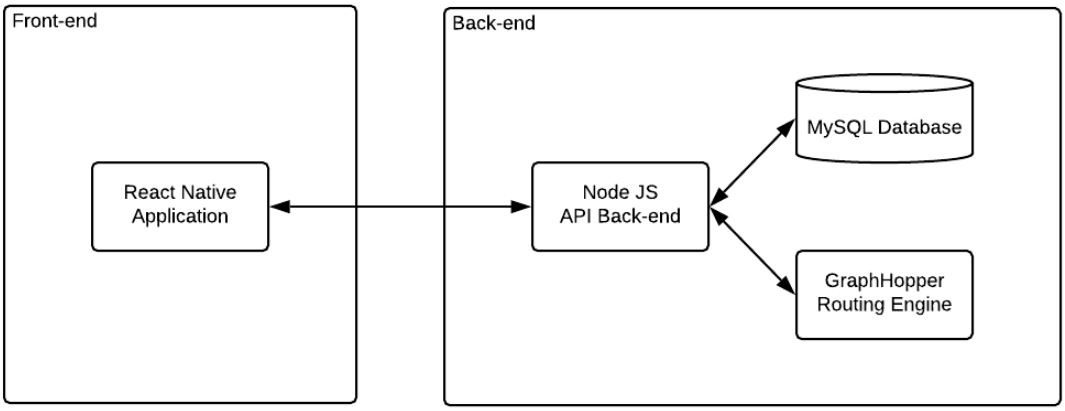


Figure 4: Overview of system components

The structure of a separate front-end and back-end is used since it is already in place at Relive and allows for a modular design. A separate front-end means that the application is potentially usable offline as well since it can be downloaded onto the device and work with offline caches. The routing engine will also be separated into its own system since it will not share any responsibilities with the web server and can not be run on the Node.JS server, it will consist of merely a Java server. Furthermore, the decoupling increases control over the system by allowing the systems to restart and crash separately without them affecting each other. This decoupling also allows for the swapping or addition of modules such that for example a development routing engine can be used.

4.3.1 Front-end Application

The front-end will allow users to interact with the application, this will be referred to as the 'client-side' of the application. The client-side of the application will be implemented, as discussed in subsection 2.3, using React Native and TypeScript. The Route Planner will be accessible for the user as both a tab in the bottom bar of the Relive application and from a button placed in Relive's Record screen. From this screen, the user will be able to start the creation of a new route. The user is then able to add waypoints to a map, implemented using `react-native-maps`, which are subsequently sent to the back-end, together with the user selected activity type. The back-end will return a calculated route between each marker, which is then displayed on said map using a `Polyline` (a polyline is explained in section 4.3.3). The user will be able to save each created route and have an overview displaying all created routes. Each saved route will also be able to be loaded into Relive's Record screen using the `Polyline` returned by the back-end.

4.3.2 Web Server

The web server needs to communicate with the front-end, routing-engine, and database to fulfill all the given requirements. Furthermore, the API will be implemented in the production server of Relive. The following set of design goals were set-up beforehand.

Security
All API endpoints should be secured in such a way that a malicious attacker can not access someone else is resource without their permission. To ensure this, the following points are dealt with:

- Validation: incoming requests with data from the user should be validated and sanitized.
- Authentication: the identity of the users behind the incoming requests should be verified.
- Authorization: users should only be able to access their own planned routes (or routes that other users have given permission to share).

It is convenient that Relive already has security in place, which automatically protects new endpoints. Instead of writing duplicate code, the same security is used to protect the routing endpoints.

Performance
The API should be designed in such a way that it supports a large number of requests, considering the size of Relive's userbase. The following roadblocks should be taken into account:

- Database query time: query times on databases should be fast.
- Route calculation time: the calculation time on the calculation of the routes should be fast.
- Purpose-oriented usage: the use of the API should only be for the given purpose, namely planning routes inside the Relive app. Measures should be taken to prevent third-parties from potentially using the API to make their own services.

The front-end and back-end will communicate over HTTP using the following API specification.

Action	**HTTP Verb**	**URL**	**Payload**
Create a planned route	CREATE	/route-builder/routes	A Route
Retrieve a planned route	GET	/route-builder/routes/:id	-
Update a planned route	PUT	/route-builder/routes/:id	A Route
Delete a planned route	DELETE	/route-builder/routes/:id	-

Table 4: API Design of the planned route resource

This specification is made using the CRUD convention[15] since it the most established convention for handling resources on the web and already in use within Relive. The actual business logic that handles authorization, validation and calculation of routes will be hidden away behind this interface.

In its most basic form, the payload the end-user will be supplied to the server is a list of waypoints as specified in the problem definition (subsection 2.1). The back-end then uses this payload to perform the following tasks:

- Take the payload and calculate a new route: e.g., a new list of geolocations that will navigate the user through the waypoints.
- Handle the persistent storage of these calculated routes.

[15] https://en.wikipedia.org/wiki/Create,_read,_update_and_delete

4.3.3 Database

The database needs to store the planned routes such that users can access them from any device. A planned route consists of several properties, which are shown in Table 5. The database of Relive will be used with a new table named `planned_route`.

Table 5: Properties and usages of the planned route table

Property	**Usage**
User	The user the planned route belongs to.
Polyline	Stores all points that are needed to show the entire line of the planned route on a map.
Waypoints	Stores all points that the user made in the route planner.
Metadata	Chosen name of the route, distance and elevation. Can be extended without altering the database table.

These properties are the bare minimum to store in the planned route table. In order to link a planned route to a user, the user id needs to be stored as well. In addition, the planned route needs an identifier, which will be used to query on the planned route table. And finally, dates about the creation and the last update of the route need to be stored.

The polyline of a route contains all coordinates of the route. On the database, this is stored using a special encoding, called the polyline encoding[16]. This encodes the array of coordinates into a string of characters that represent the coordinates. This polyline encoding simplifies the storing of the coordinates by simply needing to store a single string instead of storing all coordinates.

4.4 Design Modifications

During the course of the project, there was a pivot point in the organization of the requirements. This was due to an extensive analysis and a deeper understanding of the problem. An important factor was the segmentation of the target audience as seen in Figure 5. From the meeting with the product manager on May 14th, the target audience shifted to the 'Not Using Route Planning App' group. This is the group that makes heavy use of the built-in Record functionality but does not use a route planner app next to it. This group is known for mainly using the Relive app. The hypothesis is that this group has the biggest advantage of improving the Record functionality of the app. The Record functionality is improved by allowing users to plan a route and follow that route during the recording of an activity.

[16]Polyline encoding: https://developers.google.com/maps/documentation/utilities/polylinealgorithm

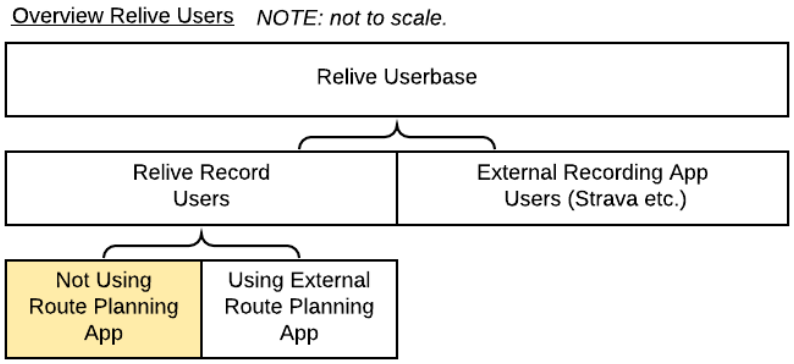


Figure 5: Overview of user segmentation for Relive Users

The shift of focus is based on the following observations:

- To correctly prioritize requirements for an application used by end-users, the user segment must be as specific as possible.
- Relive currently experiences the divergence of two segments: power users who make use of external applications and Record users who make solely use of Relive.
- The external route planning group containing power users is hard to capture and satisfy since they need to stop using their current application and switch over. This conversion problem requires to implement advanced features which their current applications support.
- The Record users without a planning do not have this conversion problem. Furthermore, since the segment consists of users who do not use a solution, a more simple and novel approach can be taken.
- The Record users within Relive are considered active and represent a large part of the user base. Retaining those users and increasing the retention rate is useful.

4.4.1 Requirement Modifications

The shift in user group led to a shift in development priority. The main implication of this is less focus on advanced features, which would mainly be appreciated by 'power users'. The features that gained a lower priority are the following:

- Routes in the route list should be importable from/exportable to a GPX file

This feature was moved down on the list of prioritization since it was concluded that this feature is used by 'power users' to import/export routes to third-party accessories (e.g. cycling trackers). This conclusion was made based on discussions with the client.

- Advanced statistic (e.g. road types & calories)

This feature was deemed too advanced for the average user, as it was concluded that extra statistics decreases the overall user-friendliness of the application.

5 Features

This section outlines the features of the application. Each screen is discussed, together with all functionality and responsibility the screen incorporates.

5.1 Route Tab

The Route Tab (Figure 6 & 45) is incorporated into the the bottom navigation bar present in the Relive app. It is created as a separate tab in order to allow easy distinguishment between the two code bases. With the two code bases being Relive's codebase and the codebase created during this project.

This tab allows the user to perform the following actions:

- Start the creation of a new route by pressing the plus-button.
- Share an already existing route.
- Edit an already existing route.
- Delete an already existing route.
- Load a route into Relive's Record screen.

The user is able to edit, delete, share or load a route by selecting the appropriate option in the dropdown menu. This menu can be opened by pressing on an already created route. This can be seen in Figure 28

Figure 6: Route Tab Screenshot

Creating a new route
When pressing the plus button at the bottom of the screen, the user gets redirected to the 'Route Builder'. More on this in subsection 5.2.

Share menu
When selecting the 'Share route' option in the menu prompted in Figure 28, the user is prompted with a share menu. This share menu can be seen in Figure 7; In this menu, the user can share his or her created route via a link on various social media. When the user presses this shared link are redirected to either the website (see Figure 44 and 12) or the Relive app, after which the shared route is imported to his or her account.

Editing routes

When selecting the 'Edit route' option, the user is redirected to the 'Route Builder' (see subsection 5.2). In this screen, the user can edit his or her already existing route. Finally, when selecting the 'Delete route' option, the user has his route removed from the database and the front end.

Loading routes into Record

When selecting the option 'Use in Record', the user is redirected to Relive's Record screen and the route is loaded into the Record map. See Figure 33. This screen was already created by Relive, however, there have been made some alterations for the project. Firstly, the route is drawn, as visible in the image, is a route planned using the Route Planner. Subsequently, the 'clear' button allows the route to be deselected. Secondly, the button in the top right corner redirects the user to the 'Route Record' screen. This screen can be seen in Figure 35 and is identical in functionality to the 'Route Tab' screen. It was chosen to make two screens, since allowing access to the route planner through the bottom navigation bar aids in users finding out the feature exists. Similarly, allowing access through Relive's Record screen eases the ability to load created routes when being out on an activity outside.

5.2 Route Builder

The Route Builder screen, as visible in Figure 6 is the main gateway to route planning. This screen allows for the actual route calculation. The user is introduced to this screen by pressing either the 'create new route' button or editing a new route from within either the 'Route Tab' screen or 'Route Tracker' screen. The Route Builder allows for the following interactions:

- Add new markers to the map.
- Edit existing markers on the map.
- Delete existing markers on the map.
- Center on user location.
- Statistics of the current route.
- Select up to 7 map-types (platform dependent).
- Select activity type.
- Name the created route.
- Show instructions.

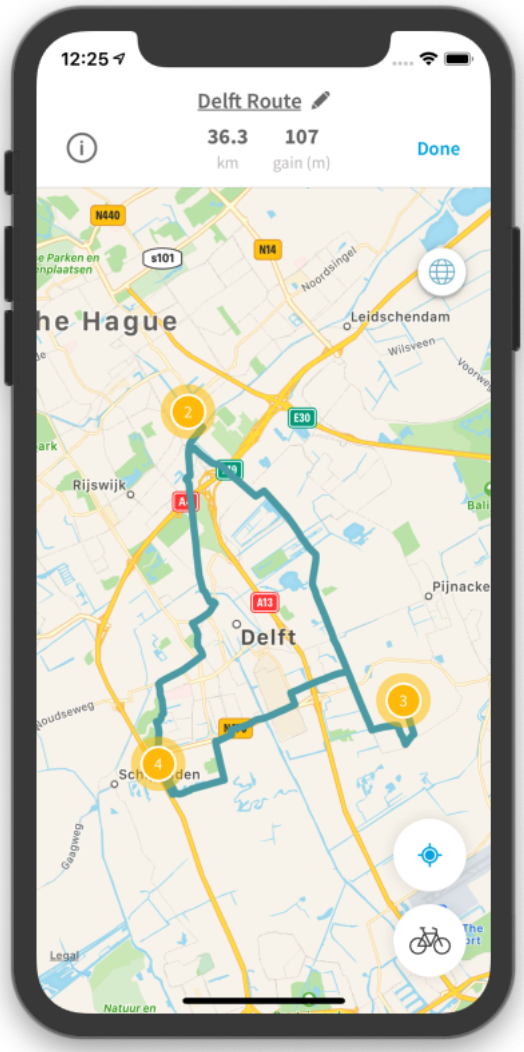


Figure 7: Route Tab Share Menu

Create route

The user can create a route by placing markers on the map by tapping it. By placing at least two markers, a route is calculated between each waypoint via a calculation done on the back-end. The route can be closed to create a loop by tapping the first marker in case at least three markers are placed.

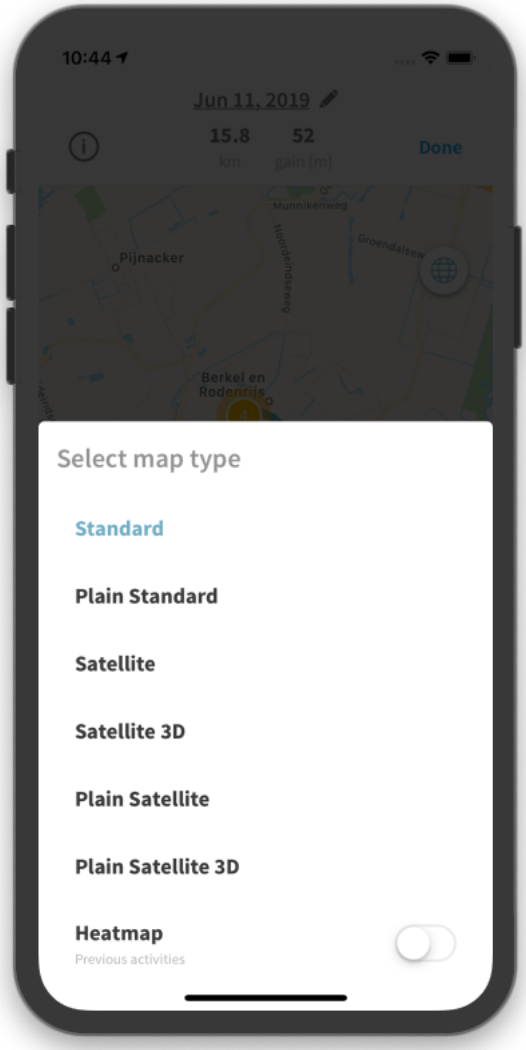
Edit route

In case a user wants to edit a route, they can do so by moving and/or removing existing markers. Moving a marker is done by long pressing said marker and dragging said marker to the desired location, after which the route is recalculated. Deleting a marker is done by long pressing said marker and dragging it into the trash bin, as can be seen in Figure 29.

Activity Type

The Relive application has support for a vast amount of activities, the route planner currently officially supports both hiking and cycling, but also allows for 'other' as an activity type which allows the user to plan any other activity that requires roads. The user can select the desired activity from the activity menu. This can be accessed by pressing on the bottom right button, see Figure 30. When an activity type is selected the route is recalculated based on said activity type. An example

of this can be seen in Figure 31. The routing engine takes different types of roads into account based on the selected activity type. E.g., when cycling paved roads are preferred over their unpaved variants.



Map Types

The user has a vast selection of map types to choose from. The user can choose the following, as visible in Fig. 8:

- Standard: Standard map type. See Figure 36.
- Plain Standard: Standard map with fewer street/ city names and no roads. See Figure 37.
- Satellite: Satellite images. See Figure 40.
- Satellite 3D: Same as 'Satellite', but with 3D models of buildings and scenery etc. See Figure 41.
- Plain Satellite: Satellite images with fewer street/ city names and no roads. See Figure 38.
- Plain Satellite 3D: Same as 'Satellite 3D', but with fewer street/ city names and no roads. See Figure 39.

Apart from these map types, the user can also enable the heatmap. The heatmap draws the routes of all activities the user has taken in the Relive App. Via this feature, the user can plan new routes without having to travel the same route again. This can be seen in Figure 42 and 43.

Figure 8: Route Builder Map Types

Miscellaneous features

A user can also start planning at his or her location by centering to it. This can be done by pressing the 'move to location' button located at the top button in the bottom right corner. There are also instructions that are opened the first time a user creates a route or if the info button in the top left corner is pressed. See Figure 32.

A user can further personalize his or her route by naming it. This can be done by pressing the title on the top of the screen. See Figure 34.

The last major feature implemented is the ability for the user to use their created routes offline, which is, e.g., useful for when a user is hiking in an area with low coverage.

6 Implementations

The implementation of the system is largely based on our initial requirements (subsection 4.1), the design goals we set-up initially and the fact that we are working with an existing code-base. We will state our implementation details, the reasoning for these choices and the factors that changed throughout the project that lead to different choices being made than our initial design choices.

The implementation of the following parts are described in this section:

- The front-end interface implemented using React Native for Android and iOS.
- The Node.JS API facilitating the connection between the front-end, the database, and the routing engine.
- The MySQL database storing the data.
- The external routing engine (GraphHopper) for the calculation of routes.
- The self-hosted routing engine based on GraphHopper in Java.

To understand the solution, it is important to have an overview of the implementation. This high level overview is described in subsection 4.3.

6.1 Front-end

As discussed in subsection 2.3 the front-end is implemented using React Native and Typescript as a separate project within the Relive app. The front-end is deployable on both iOS and Android and allows the user to exploit all features of the route builder. The front-end can be split up into five separate entities: 'Route Tab', 'Route Builder', 'Route Record', 'Relive's Record Screen'[17] and the website. The separate features of these screens can be found in section 5. This section delves into the implementation of each screen.

[17]Note: this screen was already created by Relive, however, has been altered as part of the project.

6.1.1 Route Tab Screen

The 'Route Tab' screen is implemented as visible in Figure 9. The main component is named `RouteList` and incorporates the route cards, `RouteListItem`. The `RouteList` is implemented using React Native's `ScrollView`, allowing for the scrolling interaction. The other worth mentioning components are the `RouteScreenHeader` and `PlusButton`. Both these components are implemented using React Native's `PureComponent`. These type of component aid performance by only visibly updating in case the shallow comparison of state variables yields a difference. A shallow comparison is that of comparing primitives and object references.
Due to the similarities between the implementations of the 'Route Record' screen and 'Route Tab' screen, the implementation details of the 'Route Record' are screen omitted.

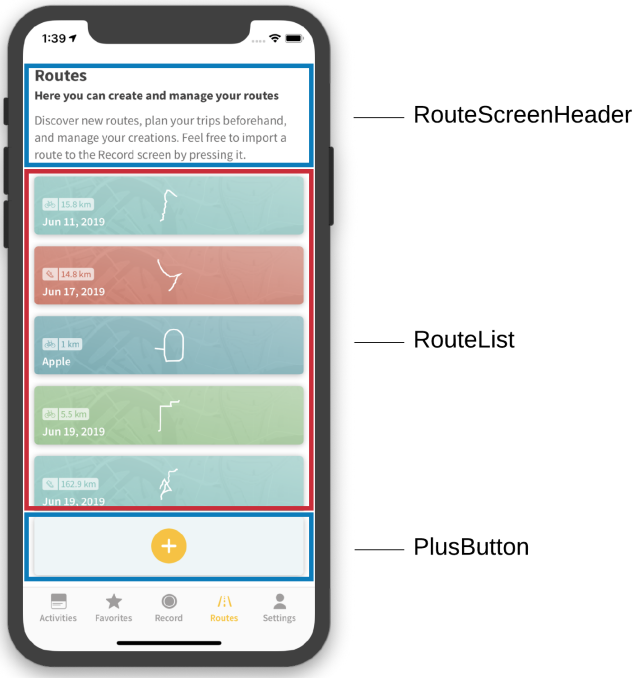


Figure 9: Route Tab Screen Outline

6.1.2 Route Builder

The 'Route Builder' screen is implemented as visible Figure 10. The main component is the `MapView` which is implemented using the package `react-native-maps` (as discussed in Section 2.3) allowing for mobile friendly map interfaces. The map consists of the following elements:

1. The map uses a `MapView` component from `react-native-maps`. On Android the Google map provider is used and on iOS the native Apple Maps provider is used. Various map types are supported which can be found in section 5 where all features are showcased. When tapping this component a `Marker` component is created.
2. Markers are implemented using the `react-native-maps`' `Marker` components using a custom `View` to create the custom look.
3. The line used to draw the route is created using a `Polyline` component of `react-native-maps` with the coordinates of the route.

The top header component `RouteBuilderHeader` contains information about the current route and is made up of the following elements:

1. Route name, clicking this route opens a popup to rename the route (Figure 34).
2. The info button opens a popup for explanation which is implemented using a modal (Figure 32).
3. Route stats shows statistics of the current route such as the length of the route and the height gain.
4. Done button to exit the route builder screen and save the route.

Figure 10: Route Builder Screen Outline

The `MapViewButtons` component consists of two different parts which are formatted using flexbox[18]:

1. Interactable buttons to change map type, go to current location and change activity type. All of these are implemented using the same button component, however, varying the icon and callback method used.
2. The delete area which is only shown when dragging a marker. When a marker is dragged to this area, the marker is deleted.

[18] Flexbox: https://facebook.github.io/react-native/docs/flexbox

6.1.3 Relive's Record Screen

The main functionality added to the Record screen is that of the drawn route. The route is drawn using the `Polyline` component. The other functionality added are the two buttons: the `ClearRouteButton`, which clears the selected route from the screen, and the `Route Planner` button, which redirects the user to the 'Route Record Screen'. Both of these buttons are formatted using Flexbox and are implemented using a `PureComponent` for the added performance.

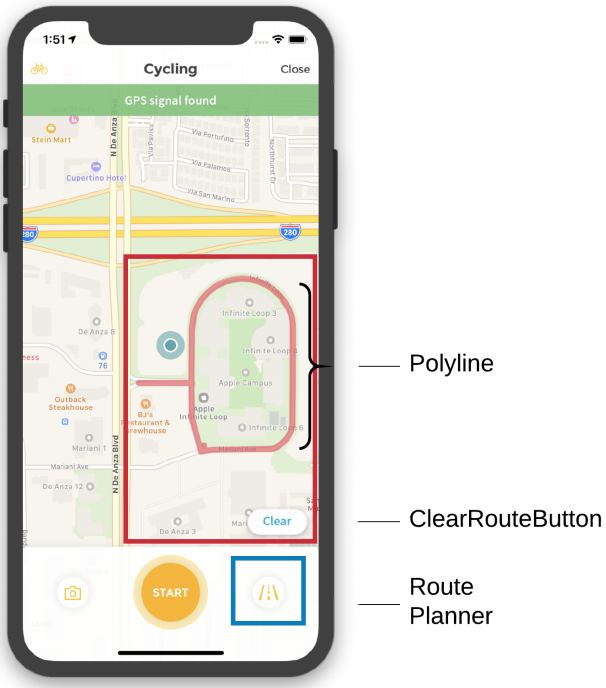


Figure 11: Relive Record Screen Outline

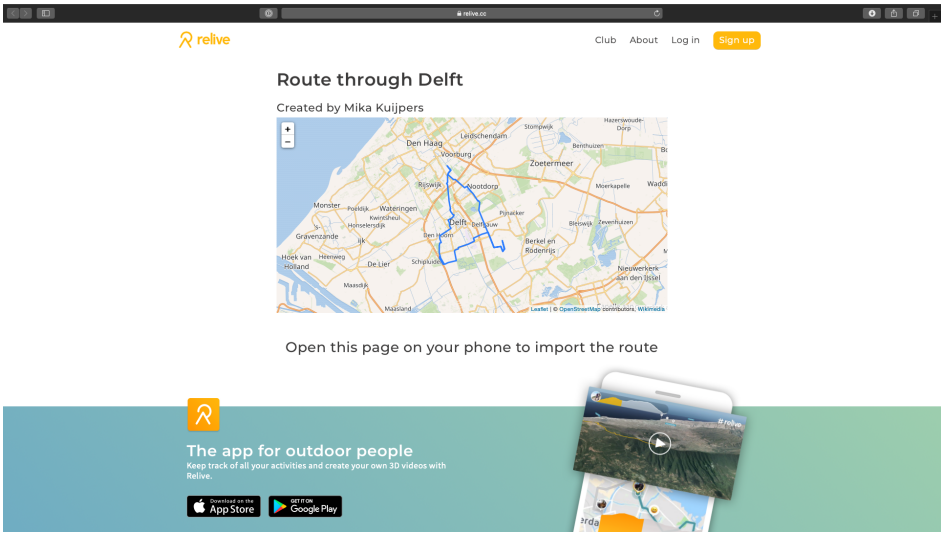


Figure 12: Route Sharing Website on Desktop

6.1.4 Website (Route sharing)

A shared route can be found on the website once a user decides to share a route. This is implemented on the Relive website using React[19], including both a mobile and desktop interface. The page contains the following elements:

1. Route title which displays the name of the route on the top of the screen. This title is fetched from the back-end.
2. Route creator which displays the user that created a route as a subtitle. This name is fetched from the back-end.
3. The map containing the route in the center of the screen to give the user a visual of what route is being imported. This route is visualized via the `Polyline` stored in the database.
4. An import button to import the route in the Relive app. On desktop a user is not able to import the route, thus a message is shown to open this page in on mobile. When pressing the import button, the user is redirected to the app through deep linking[20].

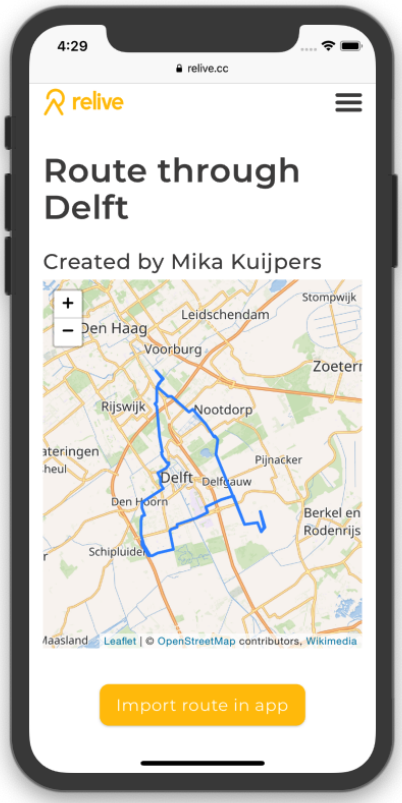


Figure 13: Route Sharing Website on Mobile

6.1.5 Offline capability

The solution supports support partial offline capability, namely being able to load an existing planned route into the tracker. This is done through a caching mechanism that makes use of the AsyncStorage API provided by React Native. On Android, this is an abstraction over an SQLite database and on iOS a serialized dictionary. To refresh the cache some sort of cache invalidation needs to be implemented and this is a hard problem since many different events such as logging out, logging in and losing internet connection can trigger many different data changes such as adding a single route and deleting all routes. Covering all these possibilities in the code can easily lead to bugs. Furthermore, the cache used is small, it only consists of a JSON payload of the routes. For these reasons, a naive approach to cache invalidation is used where om all events that change the routes of a user, the routes are cleared and stored again into the cache. This will not cause a performance problem however since the serialized routes in JSON are very small in size.

[19] Website: https://reactjs.org
[20] Website: https://developer.android.com/training/app-links/deep-linking

6.1.6 Code structure

The component-wise structuring of the main screens `RouteTabScreen` and `RouteBuilderScreen` can be seen in Figure 14 and 15, respectively. In these figures the main components, as discussed in the previous subsections, are outlined and the internal communication becomes apparent. Due to the similarities between the screens `RouteTabScreen` and `RouteRecordScreen`, has the `RouteRecordScreen` been omitted.

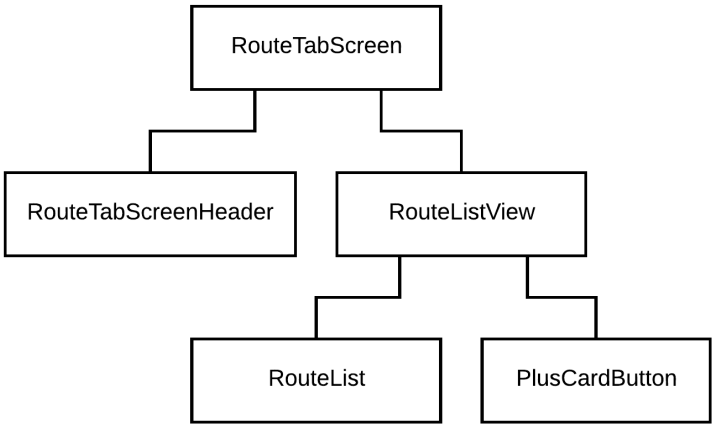


Figure 14: RouteTabScreen Code Structure

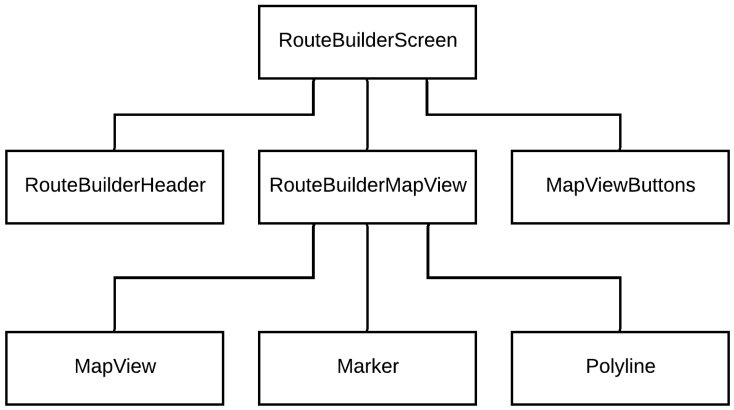


Figure 15: RouteBuilderScreen Code Structure

6.2 Routing Engine

The routing engine that is used in the application that is live in production is one using the external GraphHopper API service. The web server communicates with this service over an HTTP interface[21]. The service provides routing calculation for any place in the world for a range of activities, as discussed in subsection 2.3.3.

Additionally, a custom routing engine is implemented as a proof-of-concept, based on a routing library that GraphHopper provides. This library is available in Java and imported as a base for the project. This custom routing engine is deployed on a server within Relive but is not used in production since the proof-of-concept has the limitation that it only works on data of the city Rotterdam. The map data of the Netherlands is used from OpenStreetMaps (OSM) which can be downloaded online[22].

The team deemed setting up an instance of the engine for the entire world out of the scope of the project, since the size of the map data of the entire world is upwards of 1TB[23] and will take a significant amount of time to process. On top of that, running an instance of GraphHopper for the entire world will take upwards of 64GB RAM which the team does not have access to[24].

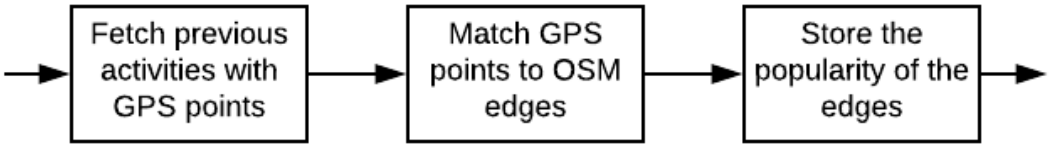


Figure 16: Batch pre-processing pipeline

Several steps need to be performed in order to incorporate popularity in route calculation. It can be viewed as a pipeline as displayed in Figure 16. The data used to produce the routes is that of Relive's Record functionality. Relive has a dataset of more than [redacted] recorded activities of users available. All recorded activities are stored in a MySQL database containing the GPS locations of the activity in a geography column.

The process of setting up the custom routing engine, based on the most popular routes, is as follows. First, a query is used to fetch all GPS locations of 100 activities inside a small area in the map (for the query, see Appendix B, Listing 1). For the proof-of-concept is the area of Rotterdam and its surroundings used. Any area can be used, as long as it is a valid location within the OSM map that is loaded into the routing engine. The second step is to match the GPS points to roads in the OSM map. However, GPS points are not 100% accurate and are not always present in the OSM map, which can cause mismatches between GPS points and roads. This is solved using map matching.

[21] GraphHopper service API documentation: https://docs.graphhopper.com/

[22] https://www.geofabrik.de/data/download.html is used in this research

[23] Map data of the whole world: https://wiki.openstreetmap.org/wiki/Planet.osm

[24] RAM usage discussion on GraphHopper: https://discuss.graphhopper.com/t/how-much-ram-to-importing-planet-osm-feb-2018/2799/2

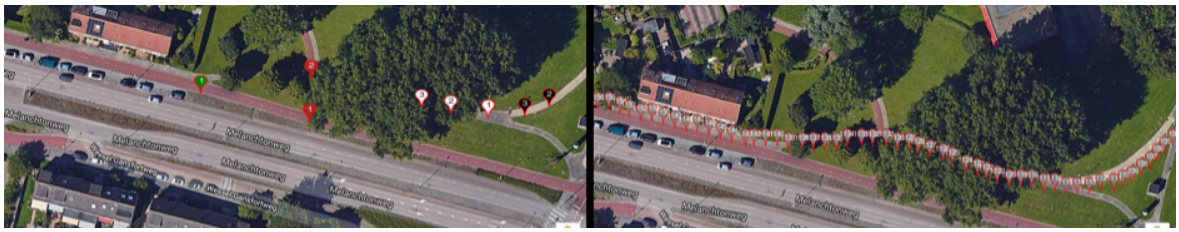


Figure 17: Badly snapped waypoints: the right side contains GPS locations, the left side the matched points.

For example, Figure 17 shows a small GPS-point-to-road mismatch. On the right side, all GPS points are displayed; in this case, the GPS points are accurate. There exists a sidewalk underneath the tree at the location of the GPS points, however, the red points on the left side of the picture show that a different road is matched, snapping falsely to the road going downwards. A small side note here is that it proved to be cumbersome to find an error since most of the roads ($> 99\%$) are matched correctly, as can be seen in Figure 18.

Figure 18: Correctly snapped waypoints: the right side contains GPS locations, the left side the matched points.

The last step of the prepossessing is to cache the number of times each edge is visited. This can be done in memory, however, this requires a massive amount of memory since there exist a vast amount of roads and activities. To achieve this, GraphHopper's memory management was extended and altered to include several extra bytes of information. These bytes are used to store the number of activities for each way. All information is cached and stored for later use in multiple files.

The goal of the self-hosted routing engine is to incorporate popularity in the routing calculation. Dijkstra's shortest path algorithm is used for the route calculations. The graph consists of edges (roads on the map) and nodes (crossroads on the map). Each edge has as weight the distance of the road. Dijkstra used bread first search to traverse the graph while keeping note of the smallest weight to each node. After the traversal, the path with the smallest weight is picked. The weight of each edge can be altered to include the activities. Decreasing the weight of an edge based on the number of activities results in an equal or smaller weight in the final path. A path is therefore more likely to be chosen in the end, if it contains popular edges. To achieve this idea, the popularity needs to be incorporated in the edges. The pseudocode in Algorithm 1 shows a possible solution. The complexity is $O(n)$ where $n = |matchedEdges|$, if *originalEdges* is stored in an efficient data structure that supports a complexity $O(1)$ for the *contains* operation. An example of this is a `hashmap`.

Algorithm 1 Popularity inclusion in OSM edges

```
1: function IncludePopularity
2:     originalEdges ← edges of the OSM map
3:     gpsPoints ← GPS points of activities representing popular routes
4:     matchedEdges ← matched edges based on gpsPoints using the GraphHopper library
5:     for edge ∈ matchedEdges do
6:         if originalEdges contains edge then
7:             popularity of edge + 1
```

The main trade-off of a popular route is the ratio between time, distance and popularity. In the test, percentages from 0 to 100 are used to decrease weights of edges based on the popularity. In Appendix A, Figure 23 and Figure 24, the distances and estimated time are plotted for each percentage between 0 and 100. In Appendix A, Figure 25, the routes of other users are displayed as a blue line. Less blue means that fewer users visited that part of the map. The main goal is to maximize the percentage since that means that the popular routes are preferred over others. However, the time and/or distance needs to be minimized, to ensure that the resulting route is not too much off the shortest path. In the image, a popular route is plotted with the percentage maximized and the distance minimized manually (see Figure 25, image 'Percentage and distance optimized').

6.3 Database

The database storing the planned routes is a MySQL database that is already in place at Relive. The planned routes are stored in a table in this database, of which the schema can be seen in Figure 19.

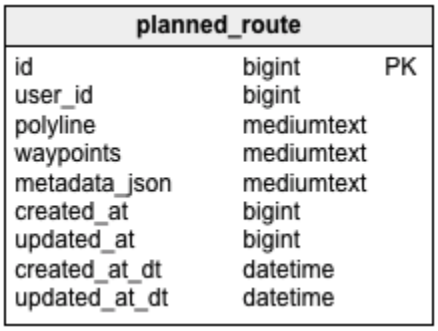


Figure 19: Planned route table

The `metadata_json` field is a special field that holds a serialized JSON object as a string. It contains the metadata of a planned route as specified in Table 6. This field is deserialized and serialized on the web server. Normally this is not an ideal solution since a database must be as normalized as possible and these metadata fields should be their own fields on the table itself. However, it allows the development team to quickly iterate and include new information in a planned route without the need to perform database migrations. This is especially important since the databases are live in production for millions of users.

Property	**Usage**
Name	The name of the planned route given by the user.
Distance	The total distance of the route.
Time	The total estimated time the route will take.
Elevation	The total elevation gain one will encounter when following the route.
Activity Type	The activity type the route is suited for.

Table 6: The metadata information of a route

6.4 Back-end Flow

The actions that follow from a user request are designed in such a way that the application feels as responsive as possible for the user as per the design goal that can be seen in subsection 4.2. While the UI can go a long way in helping realize this, the back-end should follow a model that optimizes response time. The path that a request for a route calculation takes can be seen in: Appendix B, Figure 26.

The path that a request from the front-end for a route calculation takes can be modeled in the following manner. The request is firstly handled by the front-end itself, letting the user know that something is happening through a loading icon. Secondly, the back-end receives this request and, after validation, handles the fetching of the already existent route from the database. After processing the existing route in order to determine what exactly needs to be calculated, a request is made to the actual routing engine which in turn returns a complete route. The server passes this on to the front-end so that the front-end can display the route, and subsequently, the new route is stored in the database by the back-end.

7 Testing

In order to verify the correct functioning of the code, testing of code is important. Testing reassures that new additions do not break any existing functionality without the need for time-consuming manual testing. The written tests and thoughts behind them are elaborated on in this section. The front- and back-end are tested separately and therefore discussed separately.

7.1 Web Server Tests

The back-end website, which holds the web API exposed to the front-end, is tested using a combination of unit and integration tests. The testing framework Mocha[25] is used together with the assertion framework Chai[26], due to these being the most established testing tools in the Javascript community. This is important due to the team's lack of experience testing in Javascript and the subsequent assumption for the need of online troubleshooting. The results of these tests can be found in Appendix E, Figure 51.

The main strategy to test the web server is to test every function that is directly used by an API endpoint. For every one of these functions, a scenario is created that tests whether the functions work in a regular use-case. For example, the delete route function is tested by deleting an already existent route in the database and checking whether it still exists.

The tests are more akin to integration tests than unit tests. In normal unit tests, the aim is to solely test the functionality of the unit, i.e., only testing the function that is being tested and to mock any additional functionality that is needed for that unit. For some parts of the tests however, in particular the database, there has been opted to use an in-memory SQLite database instead. This is due to the difficulty in mocking the complex database at Relive, which holds a large number of foreign key relations that need to be taken into account.

7.2 Routing Engine Tests

The self-hosted routing engine is made from the ground up and tested using unit tests. Originally, there was not set any goals for a minimum coverage percentage. However, the idea to create a self-hosted routing engine in a separate code repository was in week 5. Together with the client, it was decided that 70% coverage is adequate for this project since this allows for ensuring that the main functionalities of the project work as intended. Additionally, this percentage leaves room for untestable code (e.g., direct library calls).

In the end, the code has a 76% weighted branch coverage using Jacoco for statistics, excluding non-relevant files. The entire coverage result is visible in Appendix E , Figure 50 (the coverage includes the excluded files). Several testing principles are taken into account:

- The tests should be independent of each other.
- The tests should be fast.
- If possible, the tests should make use of the three testing phases: Arrange, Act and Assert or AAA in short.
- A test should not test too many functionalities.

[25] Website: https://mochajs.org/
[26] Website: https://www.chaijs.com/

- Anyone should be able to run the test.

Some files were excluded from the coverage results. Files were excluded if they rely mostly on other frameworks. These frameworks have tests of their own. Examples of these files are the setup of the GraphHopper engine and usage of a JFree plotting framework. Additionally, several classes containing constants are excluded.

7.3 Front-end testing

The front-end is tested using end-to-end tests executed using Cucumber[27] and Appium[28]. Cucumber is used to define test scenarios. These scenarios give an outline of what is being executed and what is expected. Appium is used to execute these test scenarios. This is a test automation framework used for mobile applications. Appium handles all interaction between the tests and the mobile simulator.

The choice for end-to-end testing is because the front-end of an application requires all part of an application to work together. End-to-end testing is a way to test the complete functionality of an application. This is something especially suited to end-to-end testing. It is important to make sure that all different parts work together well and that changes do not break the application. In that way, it also serves as a regression test since any large bugs will break these tests.

Test design strategy

The end-to-end test scenarios are designed to execute basic use cases of the route builder. Each of these scenarios covers a certain feature or step in the route builder. For example, creating a route or renaming a route. By ensuring that each scenario tests a certain part of the feature it is easy to track down bugs in the route builder. Also, scenarios that test entire aspects of the route builder are created, in order to ensure all aspects are functioning together as expected. An example of a testing scenario used can be found in subsection E.1.

Drawbacks

The team stumbled upon a couple of drawbacks during the process of testing the front-end.

By using end-to-end testing for the front-end, one major drawback was present. With major UI changes, the tests will fail since the tests can't account for major UI changes. This asks for test corrections when these changes take place. Ideally, this is something you want to avoid when creating tests. However, with end-to-end testing of a user interface, this is hard to avoid.

Another drawback is related to the library used to simulate touch input. It was not possible to simulate click events at precise coordinates on the screen. This was due to a limitation in the library. In the timespan used to make these end-to-end tests, it was not possible to find a solution to this problem. The click simulations would only click in the middle of a certain element which asked for some clever workarounds to make sure tests would work.

Another drawback or bug in the library used was that sometime a click event would not trigger. This would cause test events to fail while the app was functioning as expected. This happened infrequently but still is something that makes the tests less stable.

[27] Cucumber: https://cucumber.io
[28] Appium: http://appium.io

8 Validation

To ensure that the solution is on the right path and can be iterated upon, many processes have been used to validate the current implemented solution for the problem. This is one of the key points highlighted at Relive, which highly focuses on iterating over user feedback and analytics to ensure the solution solves the user's problem. Additionally, validation can also be performed through more technical metrics. In this section, the methods used will be explained, accompanied by their results and consequences. Additionally, the validation of the initial requirements will be validated.

8.1 Retention rate analysis

Relive has a system in place to analyze the influence of a new feature on the retention rate. First, an experiment is created with two groups: a test and control group. Both groups are similar or equal in size. The test group has access to the new feature while the control group has not. This allows for an analysis of the impact of the new feature by comparing statistics of the test and control group.

For the route builder, the experiment started in week 5 and ended in week 9. Only new users are included in the experiment, as per request by Relive. Over those 4 weeks, 0,7% of the test group is still active and 0,64% of the control group. The percentage of active users represents a metric for retention. Unfortunately, due to the short duration of the project, there is no long term retention analysis available.

8.2 A/B testing

The route builder is a new feature and releasing it to all users immediately is a bad idea. If any bugs are undetected, it is possible that the route builder breaks the application. Therefore, it is important to not release the route builder to all users. This was done by applying A/B testing while releasing the route builder feature. Within Relive, an experiment infrastructure was already set-up which allowed for easy A/B testing.

The route builder is a completely new feature for Relive, therefore the decision was made to only release it for new sign-ups. The reasoning behind this was that existing users already have a certain expectation of the Relive app. By removing the variable of user expectation, more objective feedback can be gathered. The feature was released only to new users to ensure that existing users would not see any changes. For the first release in week 5, 20% of new users were added to the test group. This resulted in around 100 to 200 users to be added to the test group per day. The following week 100% of new users were added to the route builder test group, resulting in $\pm$ 800 new users to be added per day. As of week 9, around 22.000 users were present in the test group.

8.3 Event driven validation

To validate the hypotheses set-up during UX design, the user behavior is tracked using an event-driven solution. Events are set-up on all front-end elements such as creating a route, selecting a route, et cetera. These events are stored in a database, allowing for future processing and analysis.

The information gathered from this system is used to identify whether there any bottlenecks in the UX flow. This could indicate that users are having trouble with a certain feature or certain steps are not as intuitive as expected. A snapshot of the event funnels can be seen in Figure 20 and 21.

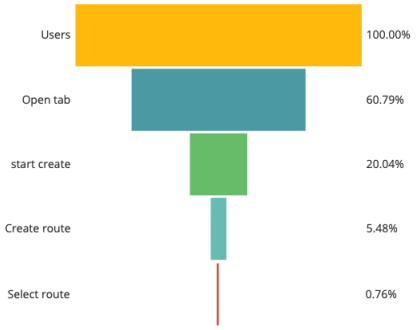


Figure 20: Events funnel release 1

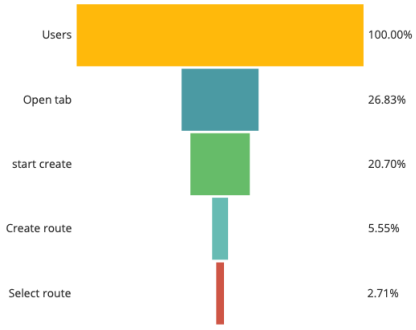


Figure 21: Events funnel release 2

Results

The event funnels show the percentage of users that have interacted with a certain feature or step at least once. Both of these event funnels contain data of around 22.000 test users. The steps shown in the event funnels are opening the route builder tab, starting the route creation process, actually finishing route creation, and selecting the route in Relive's Record screen.

The end goal of this event funnel analysis is to increase the number of users that use the route while doing activities. This corresponds to the percentage of 'Select route' in the event funnels.

From Figure 20 can be concluded that for every step in the events funnel the number of users is around a quarter of the previous step. So at each step in the route builder, only a quarter of the users proceed to the next step. There are two major things that can be concluded from this events funnel: both creating and selecting a route is not intuitive. Changes were made to improve the UX in order to improve the number of users that select a route, these changes are documented in subsection 8.5.

After the second release, which included these improvements, a new events funnel (Figure 21) was created to analyze the impact of these improvements. An optimal result would be increased 'create route' and 'select route' percentages.

What immediately can be noticed is that the 'Open tab' percentage has decreased from 60.79% to 26.83%. The cause of this decrease was not clear and was believed to be an error in the event tracking or query system since the tab had not been presented differently.

Both start route and finishing route creation had a marginal increase. What can be concluded from this is that the introduction pop-up helped but not as much as anticipated to make a significant impact. This could have been due to the longer instructions used in the introduction pop-up, shorter instructions might have been less intimidating for new users.

Finally, 'select route' saw an increase from 0.76% to 2.71%. This is the desired results since it shows that the changes made to the 'Route Record' and 'Route Tab' screen improved the user flow.

8.4 User Surveys

The users that tested the route planner were asked to fill out a short survey. The questions of this survey can be found in Appendix F. The main goal of this survey was to gain insights into the usage of the route planner and try to further interpret the results gained from event tracking discussed in subsection 8.2. In case the user indicated not having used the route planner (Figure 52), they were asked to explain as to why (Figure 53). In case the user indicated this was due to technical difficulties they were asked to explain what went wrong. In case the user indicated having used the route planner (Figure 22 & 54), they were asked as to how easy they found it to be used, how often they would use it (Figure 55), and as to how we could further enhance their experience.

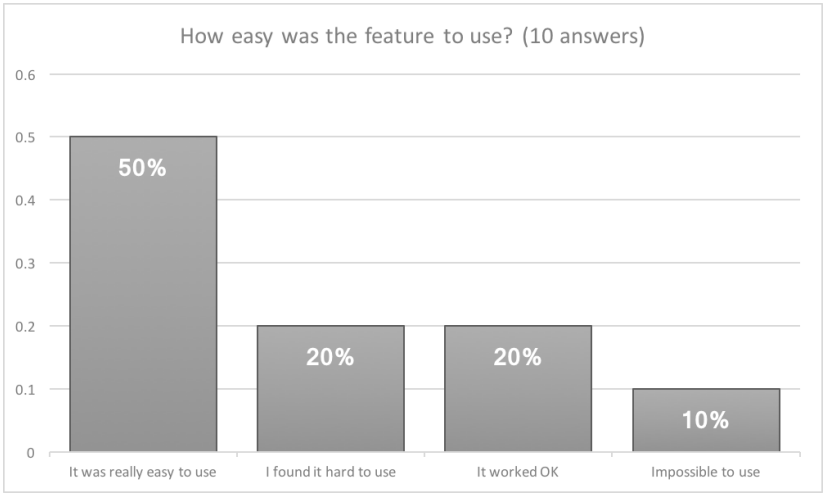


Figure 22: Survey Question 3

Results

The results of the survey are visible in Appendix F. As visible in the results, a third of the survey responses indicated as to having used the feature. Surprisingly, almost 80 percent of the responses indicate not knowing that the feature existed. This can be explained due to the fact that the users tested were new users, hence a new screen in the app would not stand out to them. Another interesting result is that no user indicated being unable to use the route planner all together, indicating that the route planner was stable to use. The range of user-friendliness varies wildly, but does lean to the side of 'easy to use'. An attempt was made to improve on this by making several changes, more on this can be read in subsection 8.5. Users also responded positively with their possible future usage, with none of them indicating as to never using it again. The open questions yielded, unfortunately, no usable information. On the other hand, the open questions answers did not indicate any complaints relating to the route calculation.

8.5 UX Improvements

Throughout the project, the UI has seen several iterations. Based on the tracked events and user surveys, there has been made a substantial change in order to improve the UX and flow to the Route Record screen. In the first iteration, the user would only be able to select a route from this screen, coming from Relive's Record screen. Based on the event tracking and surveys, the conclusion was made that this flow hindered people to load routes into Relive's Record screen. Hence, the decision was made to provide the 'Route Record' screen with the functionality of the Route Tab screen. This improvement is visible in Figure 46.

The second iteration of UX and UI improvements were made after the design meeting mentioned in subsection 3.2. The major change that came out of this meeting was that of the route cards. The route cards already had two buttons but needed a new one for the sharing functionality. The route cards were refactored accordingly to the meeting, this can be seen in Figure 47. The second refactor was that of the 'Clear selected route button', this button allowed for the deselection of a route loaded into Relive's Record screen. This button was placed in the 'Route Record' screen, however, it was moved to Relive's Record screen, as depicted in Figure 48. The last improvement that was implemented was that of the trash bin in the 'Route Builder'. This button was formerly placed on the activity change button, however, it was deemed to be unclear for users if UI components are used for multiple interactions. This button was refactored to a separate component in order to be more prominent. This change is visible in Figure 49.

8.6 Requirement Fulfillment

To conclude with the validation of the delivered product, the initial requirements will be validated and compared to the delivered project. The requirements can be found in subsection 4.1.

To summarize, all must-have's have been implemented. All of the should-haves have been implemented as well, except for some requirements related to the user interface and GPX import. This was done due to the shift in target user group and thus was not a major downside. From the could-haves, the heatmap functionality and storing routes on the Relive account have been implemented.

Considering that the must-haves, the should-haves and a number of could-haves have been implemented, it can be concluded that there is an adequate fulfillment of the initial requirements.

9 Software Improvement Group (SIG)

The Software Improvement Group (SIG) has been consulted twice during the course of the project. The code at that point in time was sent to the group after which feedback was given on said code. The goal of the SIG submissions is to improve any code quality problems before the end of the project. In this section, we will discuss the feedback that was received and what changes were implemented based on said feedback in order to improve the quality of code.

The code submissions only contain the code that was written as part of the project and not the in-house code of Relive, due to IP. Since our code is embedded in their code-base and partly makes use of their libraries, some parts of the project are not executable. SIG, however, also uses static analysis tools, hence this was not a problem. The submission is as follows:

- **Front-end (React-Native - Javascript) code-base.** This includes all the React components, business logic and tests for the front-end.
- **Back-end (NodeJS - Javascript) code-base.** This includes the API section for the route builder and the tests.
- **Routing Engine (Java) code-base.** Unlike the previous two items, this includes the whole routing engine project including its tests.

The first code submission occurred in Week 6 of the project. At this point, most of the 'must have' and 'should have' requirements were completed. The feedback can be found in Appendix I. In short, it contained the following critique points:

- Unit size that is too big.
  (This was mainly caused by routing engine code, which contained a number of script-like debug classes which greatly increases unit size.)
- No front-end test and too few unit tests.

The second point, the missing of tests, is of big importance since a lot of time was spent on front-end testing. The front-end code changes a lot, thus regression issues came up frequently. The creation of these tests would save the team a lot of development time.

These feedback points were converted to a number of concrete tasks that were completed before the second code submission, which are as follows:

- **Front-end:** Automated end-to-end testing for the front-end application by using Cucumber and Appium. This combination executes tests scenarios by simulating user interactions on a simulator.
- **Back-end:** Unit/integration tests for all the functions in the back-end API using Mocha.
- **Routing Engine:** A full test suite using JUnit for the routing engine with the target of 80% branch coverage.
- **Routing-engine:** Refactoring of the application to reduce the unit size and cleaning it to remove any debug and benchmark code.

The results of the testing efforts are further elaborated upon in section 7.

The second code submission occurred in week 9. The results of this submission can be found in Appendix I.

10 Ethical implications

The route planner has multiple ethical implications that need to be considered. Personal data is stored and should be handled in a responsible manner. In this section, the ethical implications are discussed and how these implications are dealt with.

At first glance, the route planner might be interpreted as innocent in terms of ethical implications. However, consider the following list related to planned route data[29]:

1. Health information can be deduced from the number of planned routes. How many times a person sports, together with the duration, gives a valuable health insight in the user. One of the obvious risks is that this data can be used by insurance companies.
2. Housing location can be deduced from the planned routes. Often, the planned route will start from that location.
3. Wealth information can be deduced from the number of times a route is planned without using the house location. On vacation, users plan routes in other places, which indicate how often and how long a user goes on vacation. An estimation of the costs can be made from the starting place of the planned route (e.g., the costs of accommodation or the distance from the house location to the vacation location). Together with the house location, a rough estimate could be made of the wealth of the user. Customized pricing based on the user profile can be an ethical implication of this information.

Not all implications are equally important and frequent. The house location is partly the responsibility of the user since the user decides where to plan a route. The 'personal' information might not be valid for the same reason.

All items can be considered personal information and should be handled with the necessary carefulness. The user needs to be informed about the data implications of planned routes in the privacy statement or other documentation. The usage of personal data needs to be documented as well. Additionally, the ethical implications are present, but may not be a risk if the data is not used apart from helping the user to plan and travel a route.

[29]The assumption is made that a user plans a new route for each activity. Record information of the user can be utilized to know when a planned route is traveled.

11 Discussion

During the project, several issues came to light that are up for discussion. In this section, issues, remarkable events, and memorable situations are discussed. All information is based on the 10 weeks at Relive, including both the research and coding phase.

A remarkable notion is that the project included two research phases in different weeks of the project. The first is the expected research for the route builder in general. The second is regarding including the most popular routes in the routing calculation. To increase the scientific value of the project, the team decided to take the route calculation to the next level. Unfortunately, this was in week 6 and there was limited time available. Instead of using the remaining time to solely perform research, the team decided to implement a customized routing algorithm that was likely to work, based on a minimum amount of research. Although it worked in the end, the scientific value would have been higher if more time was spent on researching instead. A small side note of the resulting customized routing algorithm is that there may be other ways to combine the two different problems. At the same time, the algorithm tries to minimize the distance and maximize popularity. Another issue might be that popularity and distance are not measured in the same units. An idea is to customize Dijkstra's algorithm in order to support two different weights instead of combining distance and popular in one.

Another thing to remember is the importance of choosing and focusing on a user group. From the start, all users were considered. The product manager at Relive indicated that issues are easier to prioritize if there is a user group to focus on.

12 Conclusion

The route builder is an extension of the current Relive application. It enables Relive users to plan a route within the app and use the route for their sporting activities. This includes a fully embedded user interface in the Relive app, an extension to the web server which delegates route calculation to a routing engine service and handles data storage, and a proof-of-concept routing engine which favors popular routes instead of shortest routes.

The initial requirements consisted of a functioning route builder and route calculation. With the additional side goal of increasing the retention rate of users.

Surveys are used to test the functionality of the route builder. The route builder is used in an experiment containing 22 thousand users. All users were tracked, analyzed and asked to fill in a survey. The tracking results indicated that 60.00% of all test users opened the route builder feature and 5.55% planned a route. A low percentage is expected since all users in the experiment are new users, as requested by Relive. From the surveys we can conclude that users responded positively with most indicating that they would continue to use the route builder.

The second major requirement is that routes need to be calculated. In the route builder experiment, Dijkstra's shortest path algorithm is used to calculate the routes. The surveys did not indicate any complaints relating to the route calculation. In addition, as a proof of concept, custom routing is explored, based on popular routes of users. Based on the results, the distance of the routes increases as the degree of popular routes usage increases. By our own observation, the custom engine is able to favor popular routes. However, the conclusion can not be drawn that the generated routes are also favored by users in practice.

Unfortunately, the long term retention of users cannot be analyzed during the project due to time constraints. In the short term, 0.70% of the test users are still active after 4 weeks in contrast to 0.64% of the control group. These percentages seem rather low, however this can be explained due to the fact that the experiment consists of new users only, as per request by Relive. There is a small increase in retention of the users that have access to the route builder feature (the test group). However, there can not be made any conclusions on the long term effects on the retention rate.

To conclude, there is an adequate fulfillment of the initial requirements. The route builder is able to be used by users in a real production environment and provides all functionality one expects from a route planner. The proof-of-concept custom routing engine is functional, however, it needs proper testing and must be deployed on a large scale for it to be usable in a production environment. It can be noted that generally speaking the 3D Route Planner project can be considered a success.

Today, the route builder is not available in the Relive app for all users, but in the future it might help you plan your sporting activities!

13 Recommendations

During the project, multiple ideas were documented that might be useful but not realistic to implement during the short timespan of the project. One of the recommendations is to use the planning tool together with the video creation functionality. People go on vacation all the time and it is inconvenient to record the entire trip if it spans several days or even weeks. These people might still value a video of the trip. The route builder enables this functionality by allowing the users to recreate the route they traveled and turning that route in a video.

The route planner can be optimized for off-road sports using previous activities of all users. Most off-road sports like winter sports, hiking or mountain biking can use the planning tool but it will lack some of the best off-road roads since those roads are not known to the map. These roads are still traveled by other users and stored in the Relive databases. Using these previous activities, the route planner can incorporate new routes to increase the support for off-road activity planning.

Although route sharing is implemented during the project, this can be extended. A recent new feature of Relive allows users to have a feed with posts of other users. Sharing of planned routes can be extended such that it is shared utilizing the feed feature.

Additionally, the route planner can be extended to increase the usefulness for other target audiences. Performance-oriented users can use more advanced statistics, such as types of road and proper time estimation.

Finally, the route planner could be extended to support round trip routes. The routing engine that is used supports round trip routes, however, these are not implemented due to the short timespan. This is something that can be added in the future as this could enhance the route planner feature for users.

References

Developer Guide | Directions API. (n.d.). Retrieved 2019-04-25, from `https://developers.google.com/maps/documentation/directions/intro`

Google Adds Trail Maps For 100 Additional Ski Resorts To Google Maps. (n.d.). Retrieved 2019-04-25, from `http://social.techcrunch.com/2013/03/25/google-adds-trail-maps-for-100-additional-ski-resorts-to-google-maps/`

GraphHopper Directions API with Route Optimization. (n.d.). Retrieved 2019-04-25, from `https://www.graphhopper.com/`

Highsmith, J., & Cockburn, A. (2001, nov). Agile software development: The people factor. *Computer*, *34*(11), 131-133. doi: 10.1109/2.963450

MapKit | Apple Developer Documentation. (n.d.). Retrieved 2019-04-25, from `https://developer.apple.com/documentation/mapkit`

Moogk, D. R. (2012). Minimum viable product and the importance of experimentation in technology startups. *Technology Innovation Management Review*, *2*(3).

Newson, P., & Krumm, J. (2009). Hidden markov map matching through noise and sparseness. In *Proceedings of the 17th acm sigspatial international conference on advances in geographic information systems* (pp. 336–343).

OpenStreetMap. (n.d.). Retrieved 2019-04-25, from `https://www.openstreetmap.org/`

Pai, N., & Li, Y. (2014, August). Pricing and competition in mobile app markets. In *2014 11th International Conference on e-Business (ICE-B)* (pp. 261–266).

Project OSRM. (n.d.). Retrieved 2019-04-26, from `http://project-osrm.org/`

Schwaber, K. (2004). *Agile project management with scrum*. Redmon, Wash: Microsoft press.

Valhalla routing engine. (2019, April). Valhalla. Retrieved 2019-04-26, from `https://github.com/valhalla/valhalla` (original-date: 2016-01-19)

Appendices

A Results

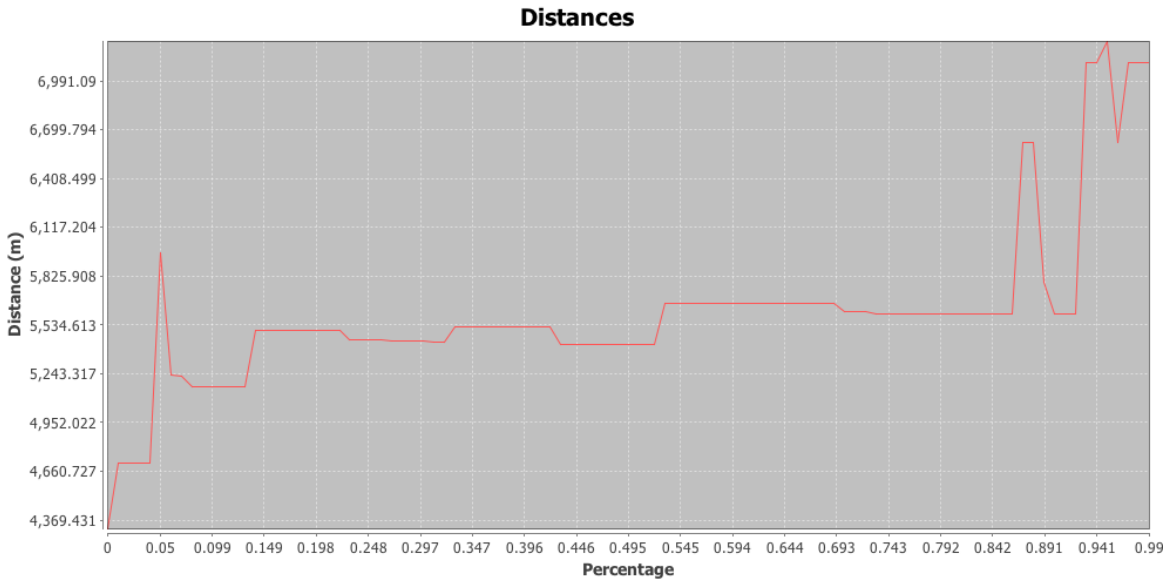


Figure 23: Customized routing algorithm results

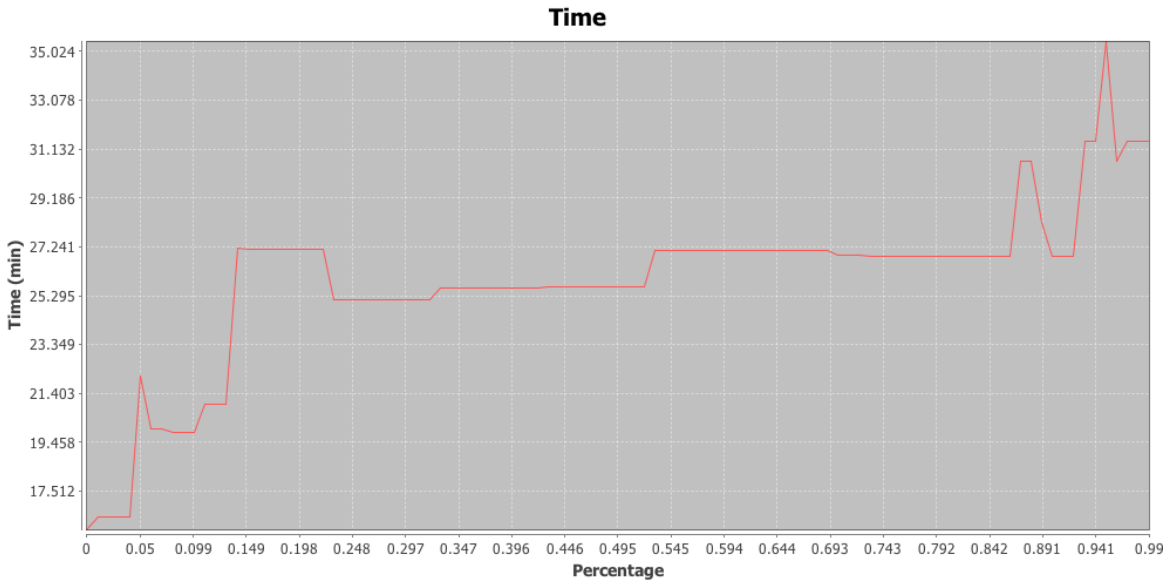


Figure 24: Customized routing algorithm results

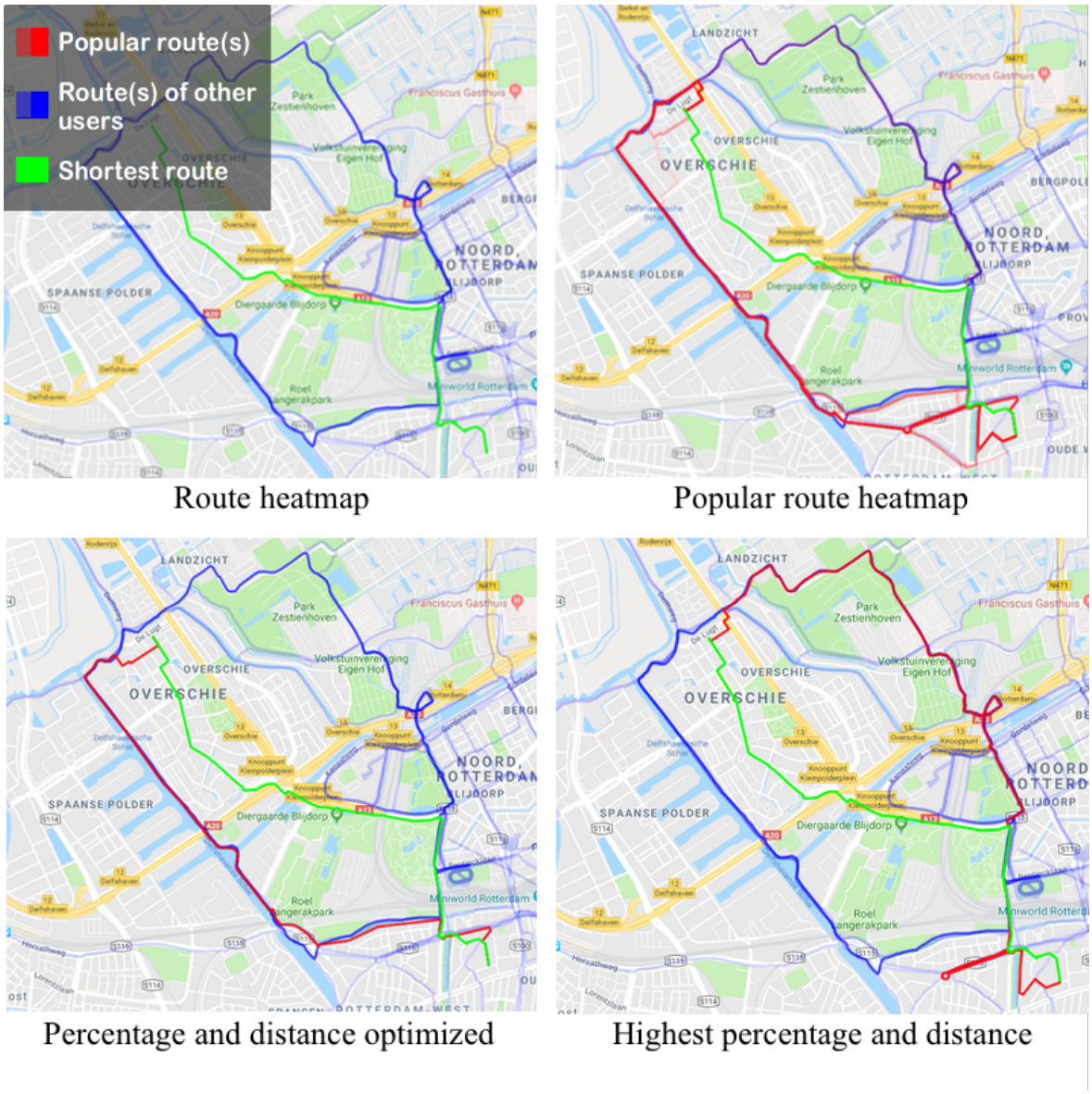


Figure 25: Customized routing algorithm results

B Implementation Diagrams

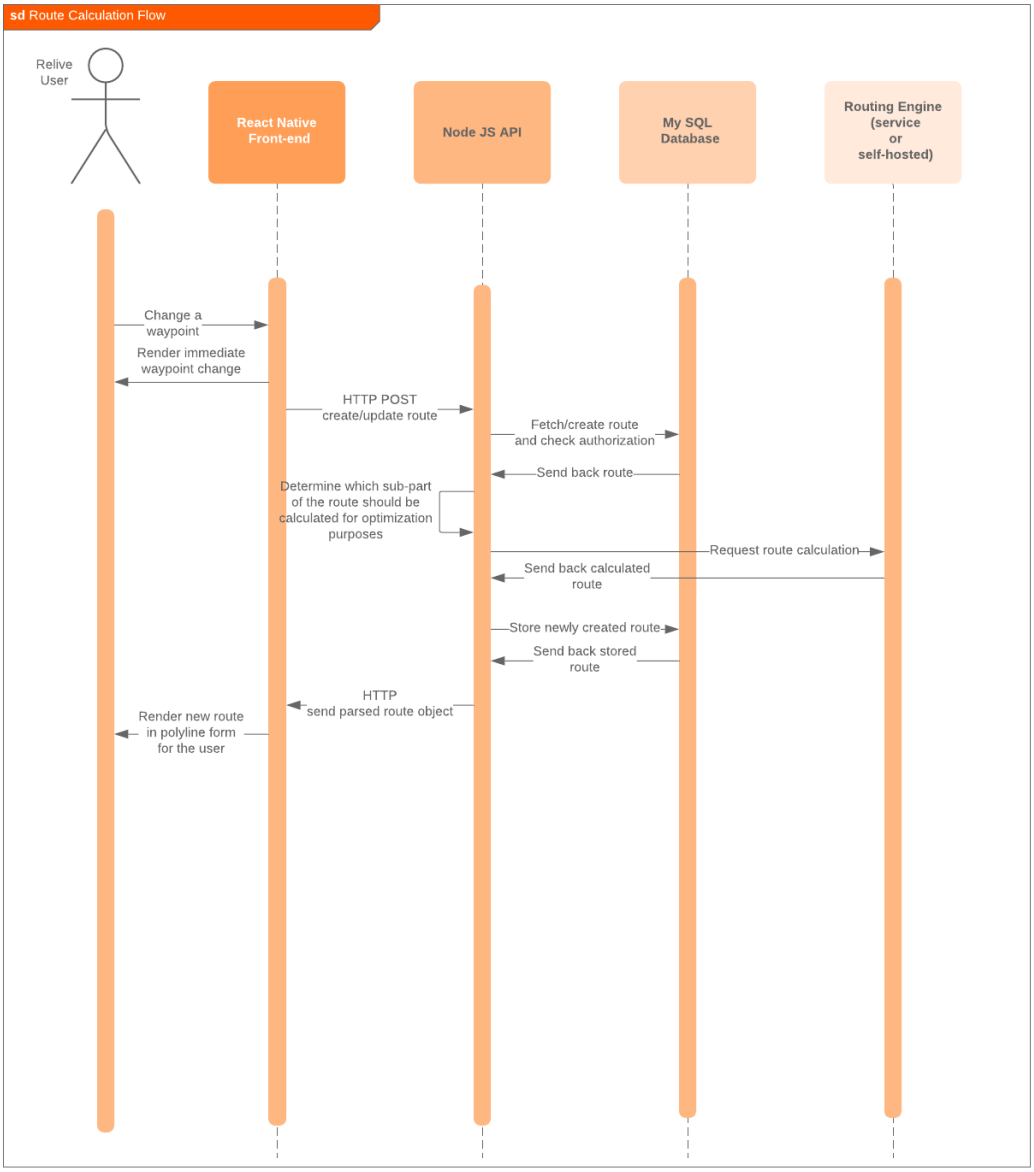


Figure 26: Route calculation flow

```sql
SELECT *
FROM route
WHERE MBRContains(
    GeomFromText(LINESTRING(4.3584268787, 51.894647852, 4.535612356, 51.991960882)),
    route.locations
)
LIMIT 100;
```

Listing 1: SQL Query to get 100 routes inside a bounding box

C MoSCoW

Must Haves

Route creation

- A 2D map which can be interacted with (e.g., zoom-in, draggable, rotatable).
- Ability to add and change start and destination waypoints.
- Waypoints must be addable to the map.
- Waypoints must be deletable from the map.
- Waypoints must be editable by users (by changing the place name manually).

Route calculation

- A route must be calculated between all waypoints after each waypoint change.
- The user must be able to indicate the activity type and must get routes generated based on said activity type.
- The distance of the current route must be visible.
- The current route must be highlighted on the map (using a colour).

Route visualization

- A route from the personal route list must be able to be loaded again as the current route.
- The route must be visible in the record feature.
- The route must be visible when offline.

Should Haves

Route creation

- Waypoints should be shown in an ordered waypoint list.
- Waypoints should be editable during route planning.
- Route should be editable during an activity.
- Routes in the route list should be importable from/exportable to a GPX file (a GPX file is a standard file format capable of saving a route).
- A route should be deletable from a personal list
- A route in the personal route list should be share-able on various social media and Relive.
- The altitude gain of the current route should be visible.

Route

- A route must be savable to a personal route list for later use.

Map

- The map should be viewable in 3D.
- The map should have a button to toggle between a 2D/3D map.

Could Haves

Route calculation

- A heat map could be visible indicating the routes of previous activities.
- A heat map could be visible indicating the routes that others have travelled.
- Routes could be calculated based on activities of other users.
- A permutation of a route could be created by varying the route between each waypoint.
- Statistics such as the different types of roads and amount of calories burned could be visible.
- The personal route list should be saved on the account of the user.

Route creation

- Point of interests could be shown on the map (such as sightseeing locations & eat/drink locations)
- The route could indicate the road types by drawing specific colors.
- The user could be alerted when he is going off route.
- Specific maps should be downloadable to support offline functionality.

Collaboration

- The planner could incorporate collaborative planning.

Organization

- The ability to organize route plans, by grouping multiple routes.

Would/Won't Haves

- Navigation instructions during activities when planned.
- Augmented reality or virtual reality.
- Marketplace to share routes.
- Automatically transform planned route into a "Relive Movie".
- Suggestions for complete routes, created by other users.
- Auto-generate routes (with parameters such as: length, preferred route type, etc.).
- Suggest alternative routes based on already completed routes.

D Feature Screenshots

Figure 27: Route Tab Share Menu

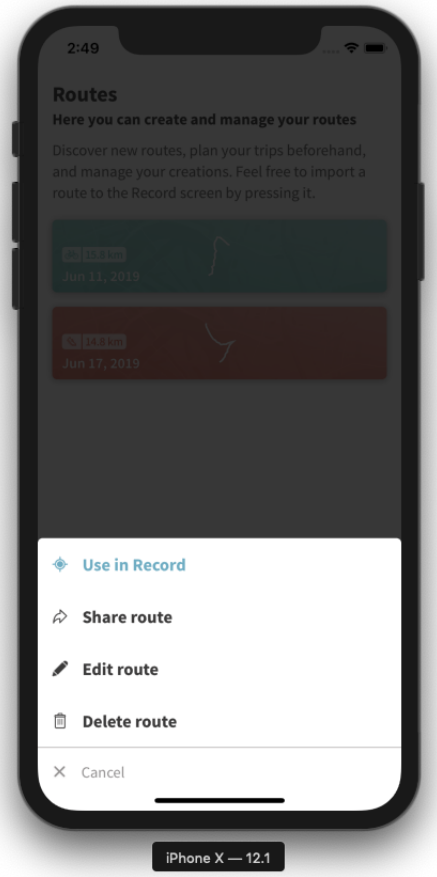


Figure 28: Route Tab Dropdown Menu

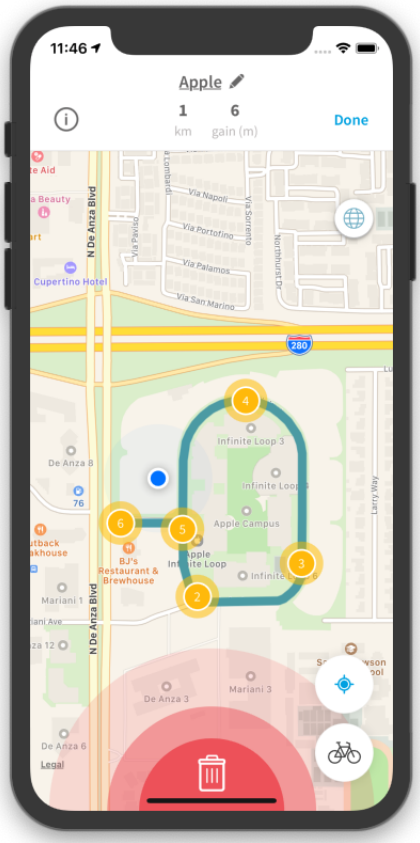


Figure 29: Route Builder Trash Bin

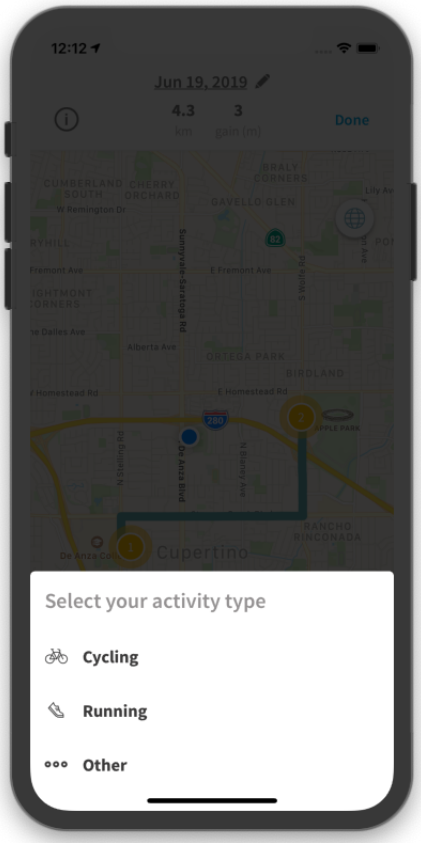


Figure 30: Route Builder Activity Menu

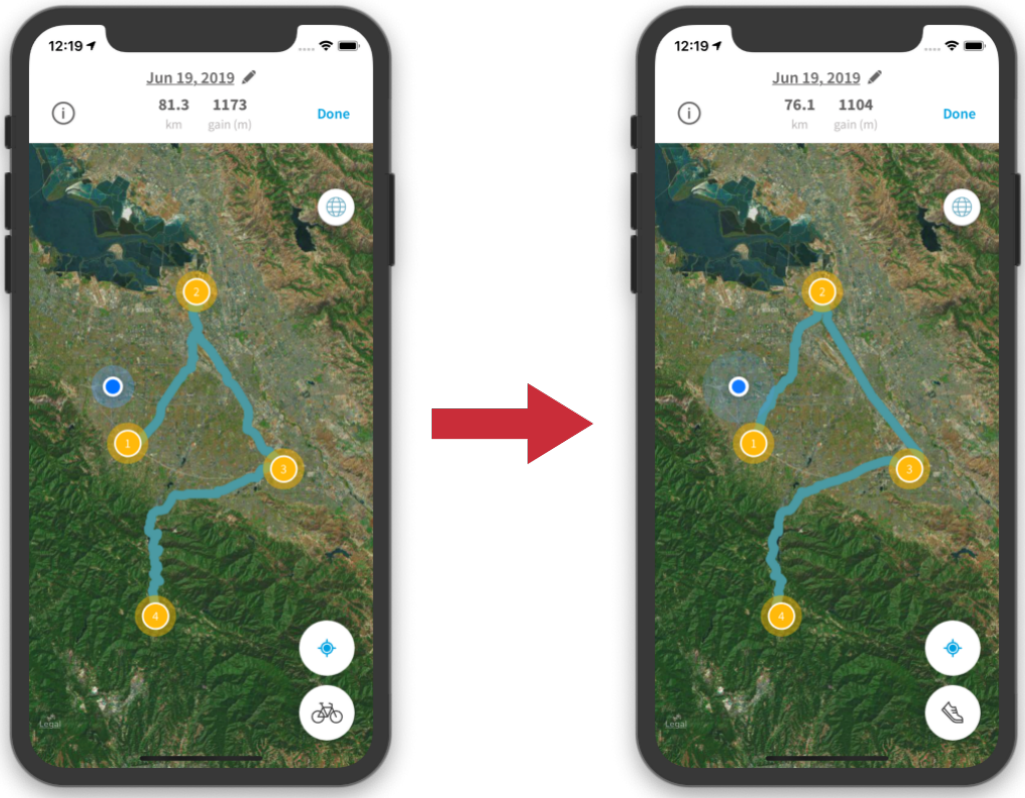


Figure 31: Route Builder Activity Change

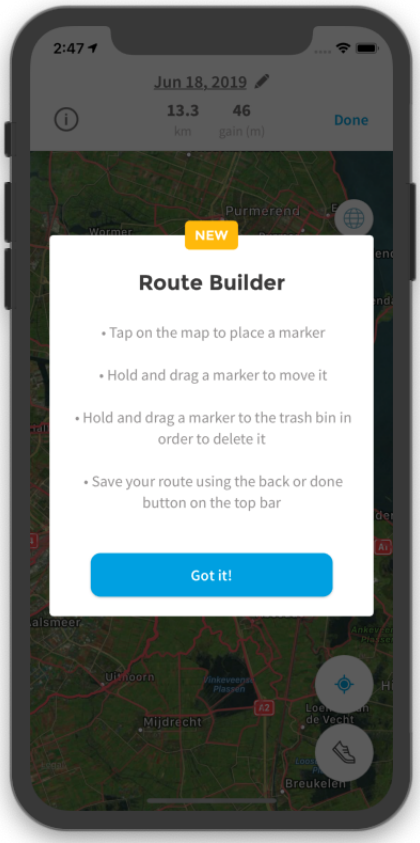


Figure 32: Route Builder Introduction

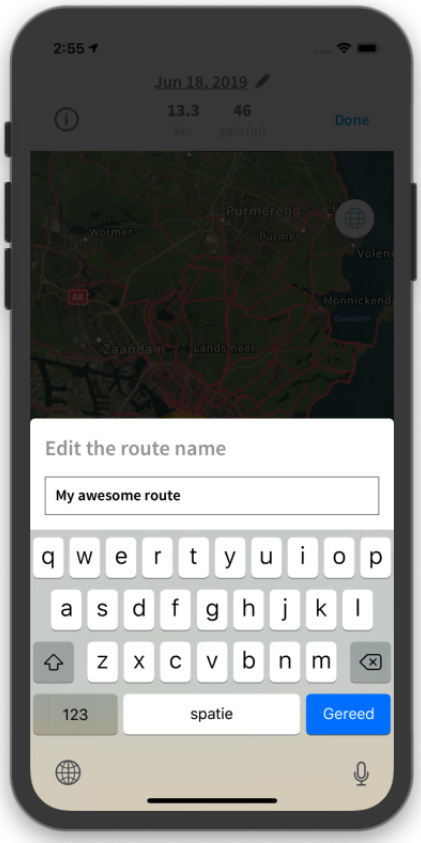


Figure 34: Route Builder Name Changing

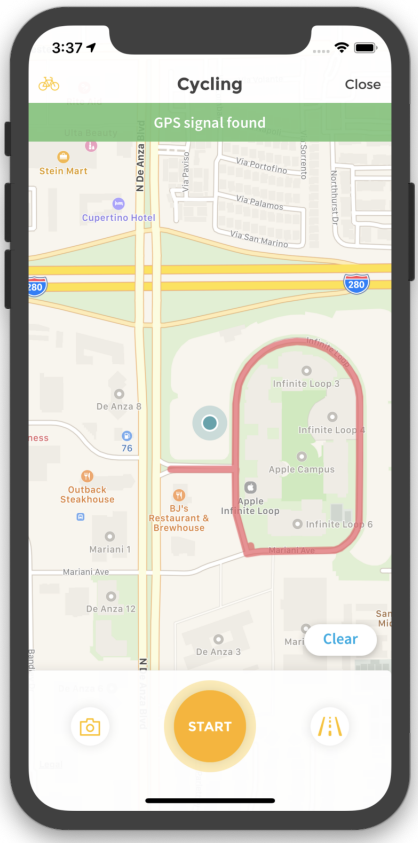


Figure 33: Route in Relive Record

Figure 35: Route Record screen

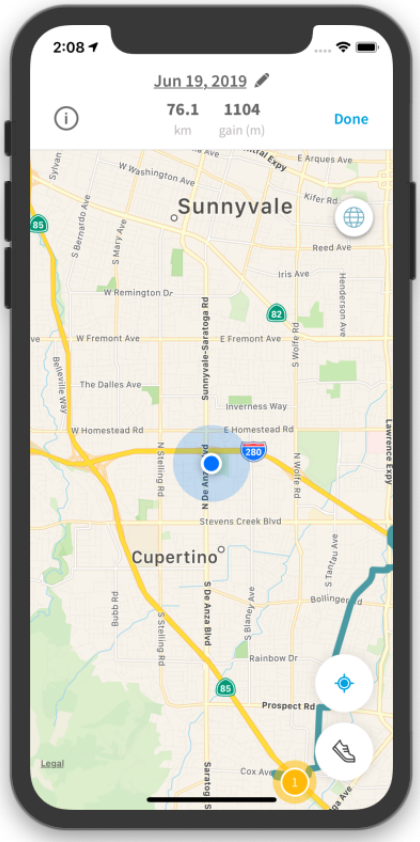


Figure 36: Map Type Standard

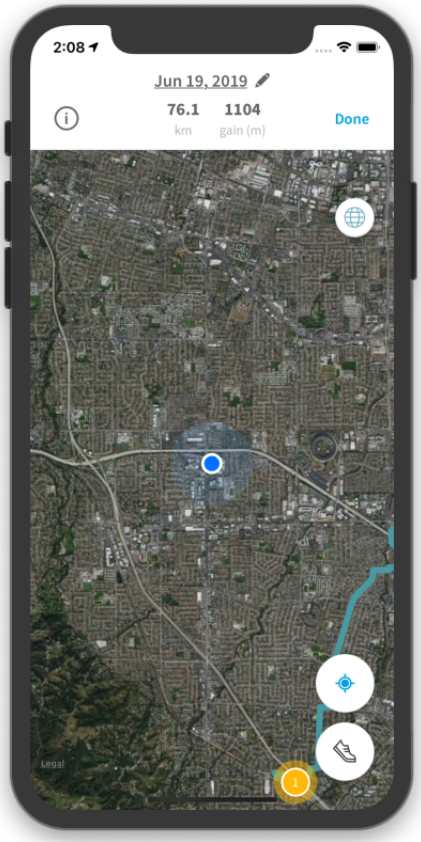


Figure 38: Map Type Plain Satellite

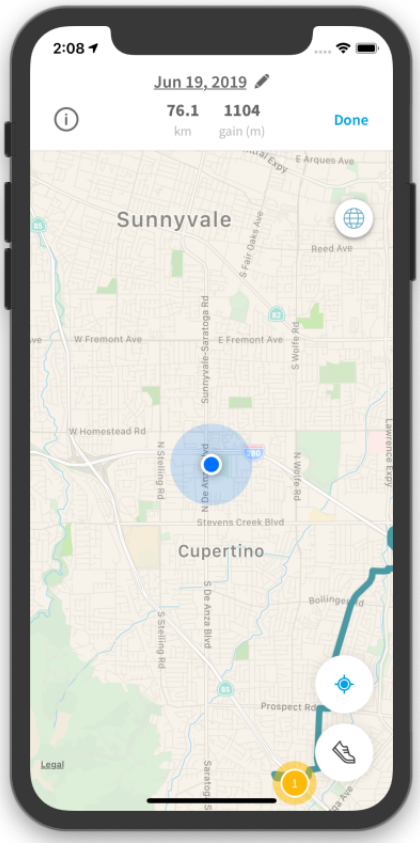


Figure 37: Map Type Plain Standard

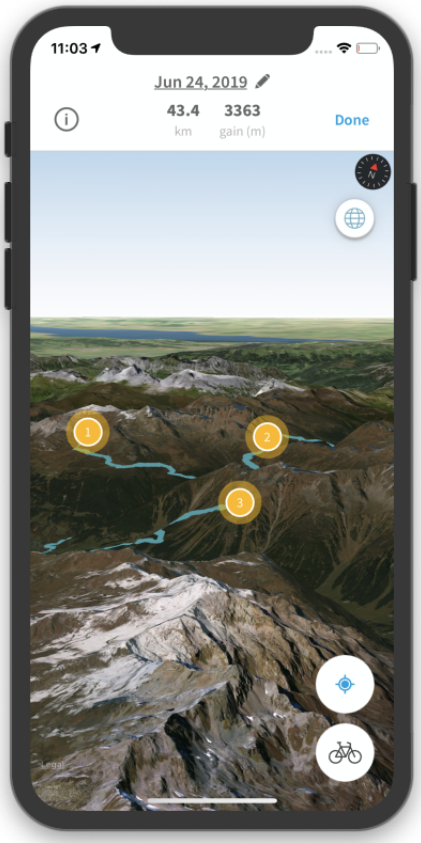


Figure 39: Map Type Plain Satellite 3D

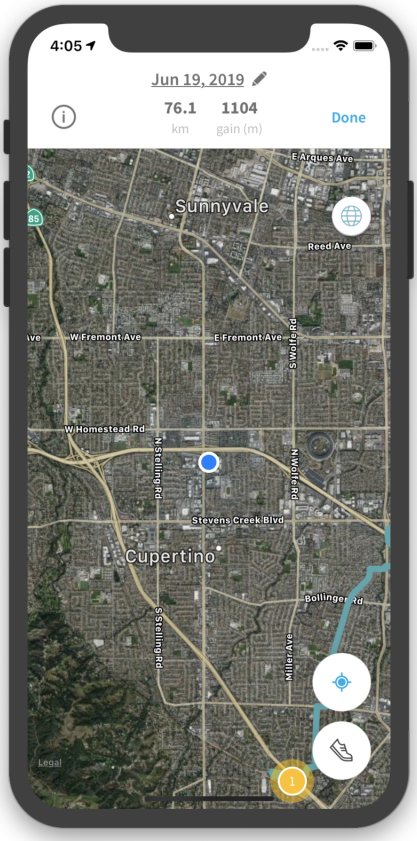


Figure 40: Map Type Satellite

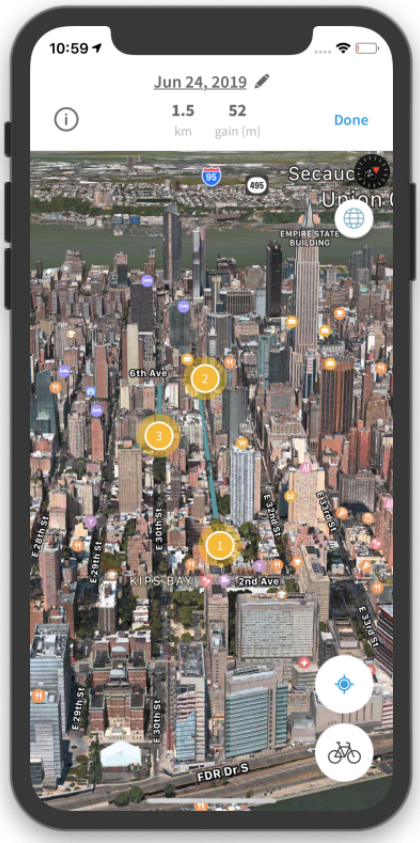


Figure 41: Map Type Satellite 3D

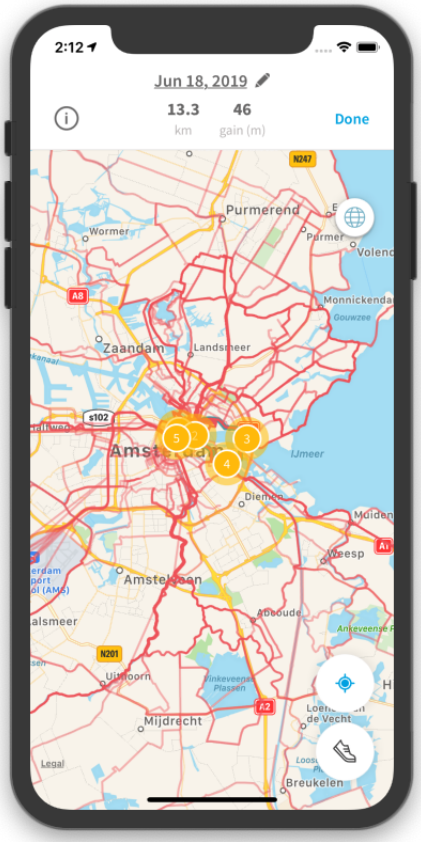


Figure 42: Heatmap Standard

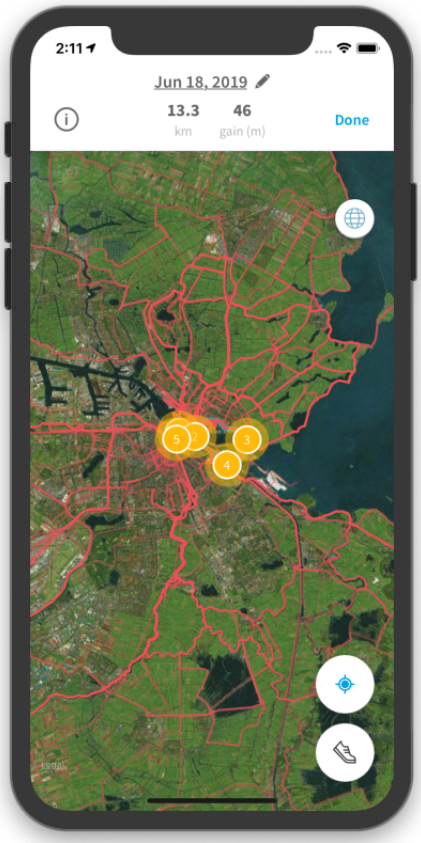


Figure 43: Heatmap Satellite

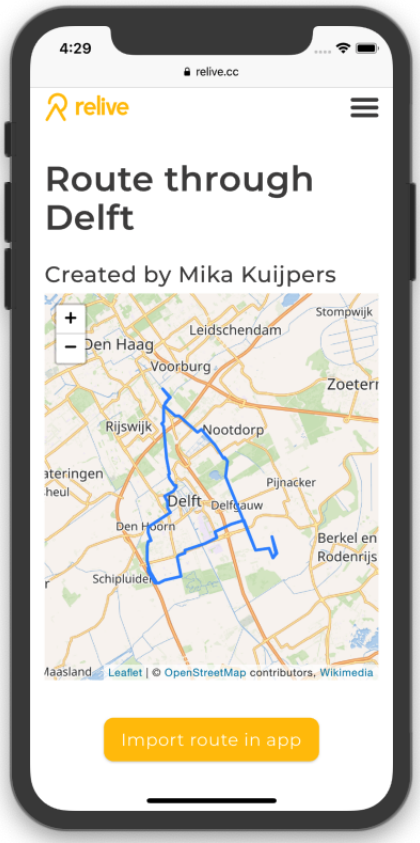


Figure 44: Route Sharing Website on Mobile

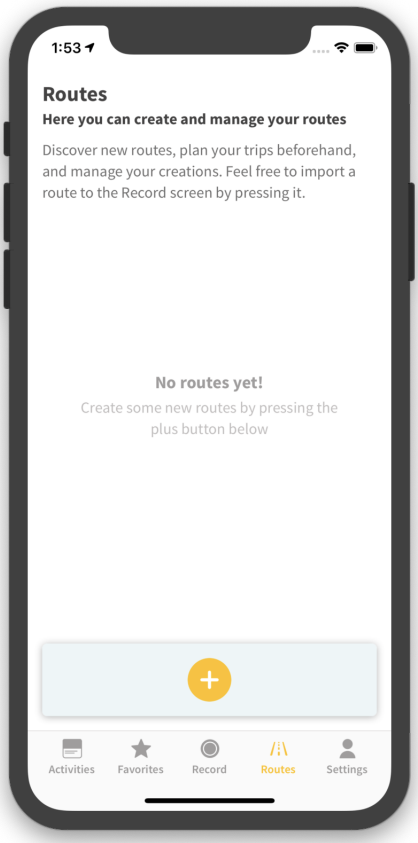


Figure 45: Route Builder No routes

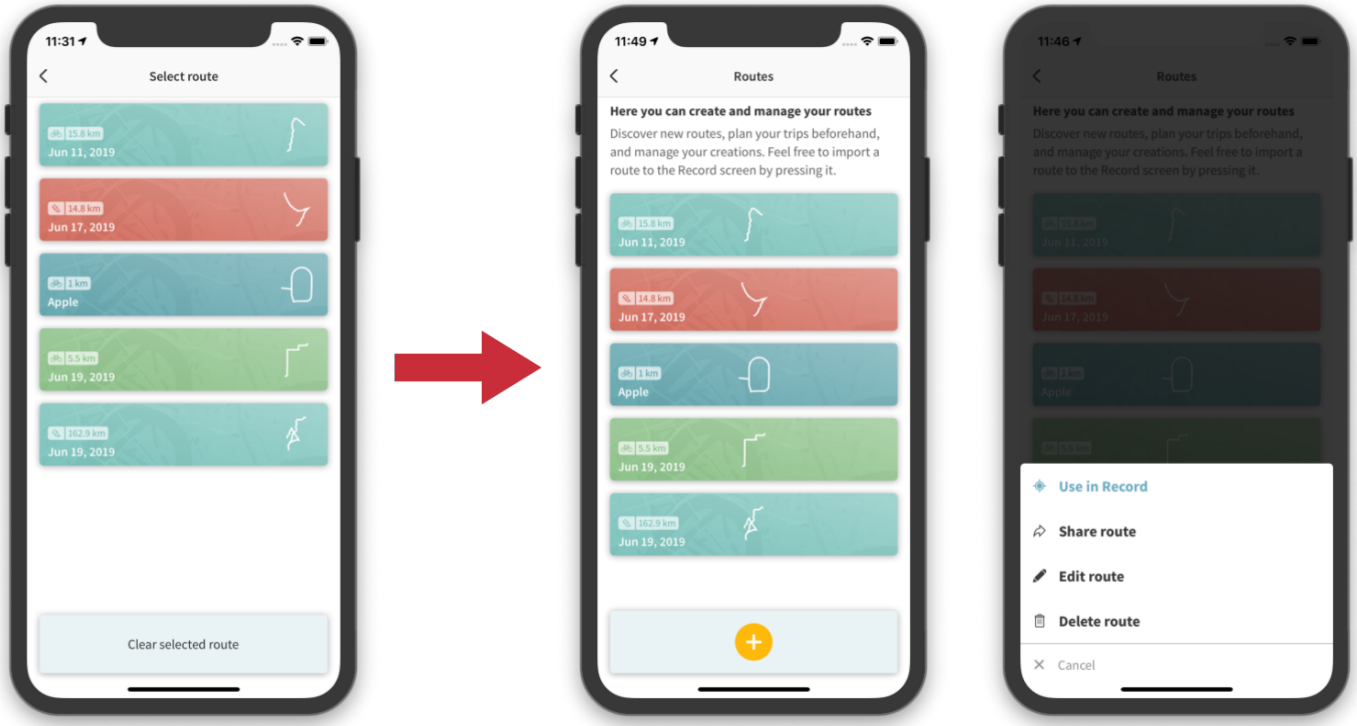


Figure 46: Route Record Improvements

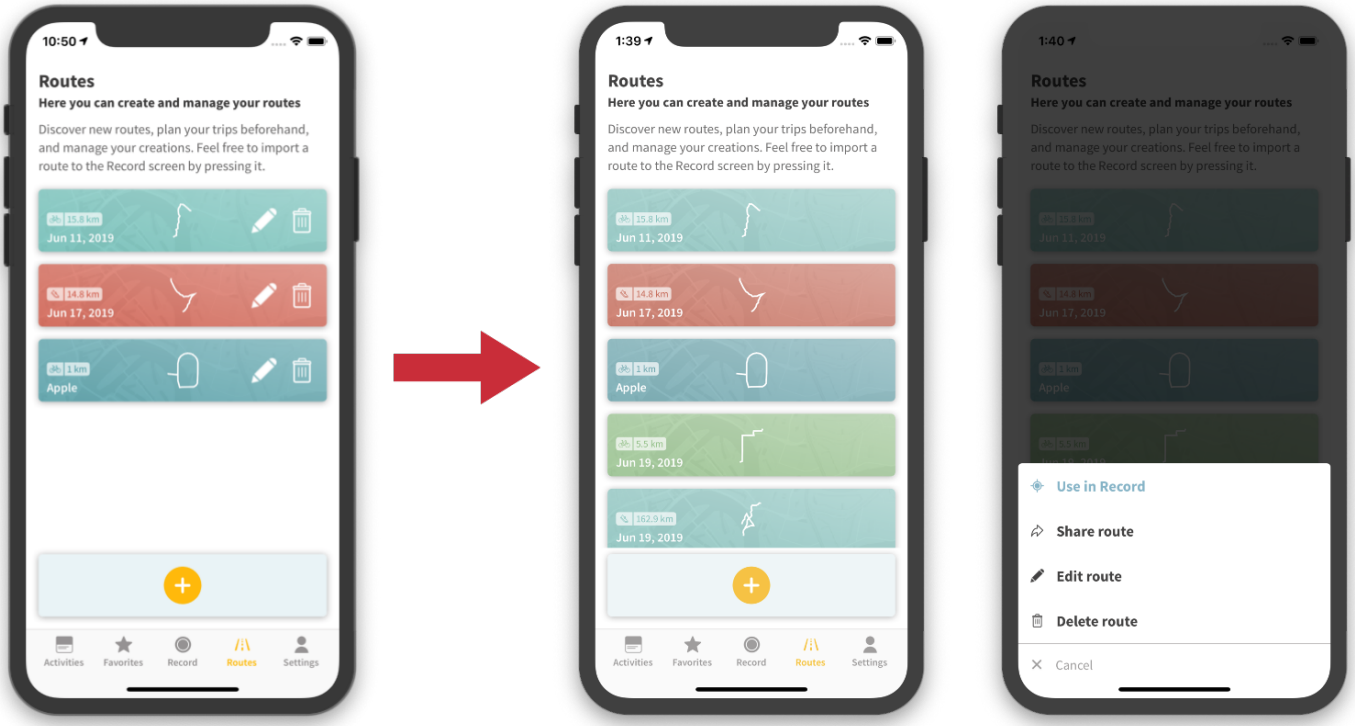


Figure 47: Route Builder Activity Change

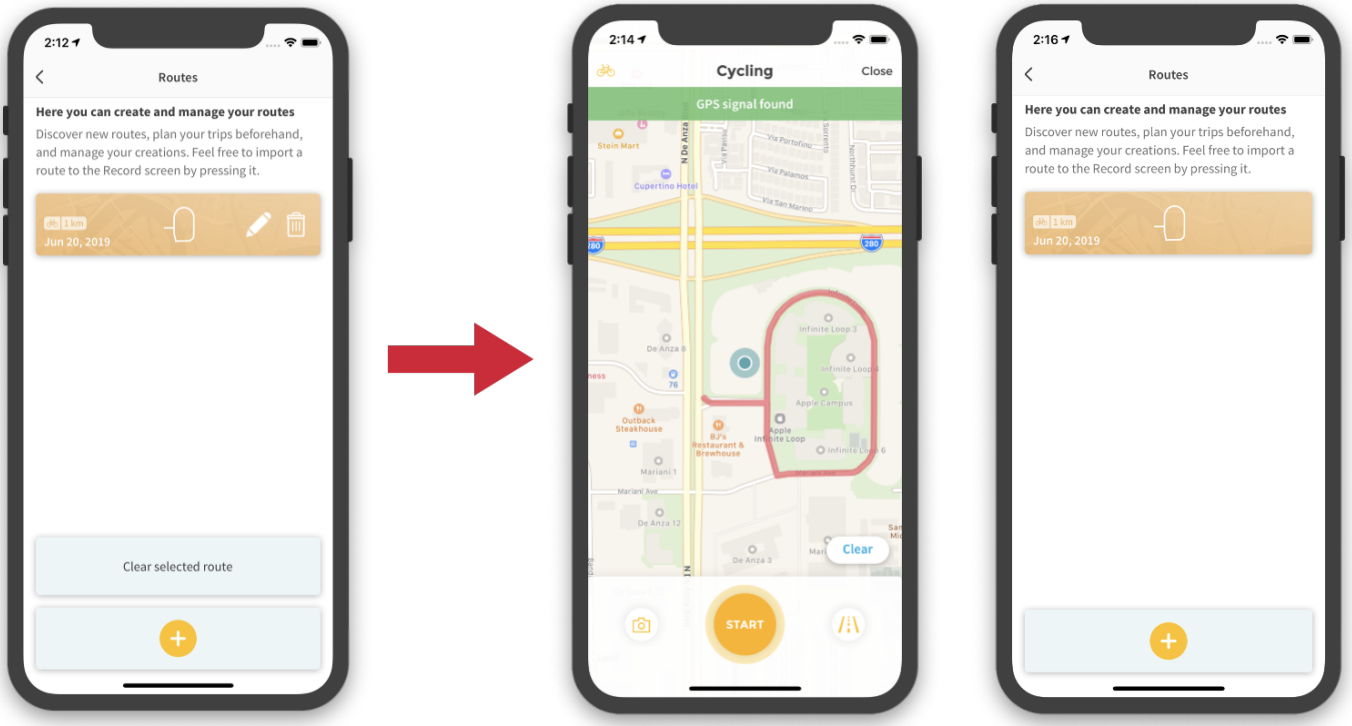


Figure 48: Clear Selected Route Improvements

Figure 49: Route Builder Trash Bin Improvements

E Testing

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
com.relive.routing.app.algorithm		21%		8%	26	31	118	137	20	25	2	4
com.relive.routing.app.main		42%		33%	31	47	87	159	18	32	3	5
com.relive.routing.app.util		91%		88%	10	45	11	116	7	32	0	8
com.relive.routing.app.models.plotting		90%		50%	10	43	1	73	7	39	0	3
com.relive.routing.app.custom		87%		75%	4	20	2	40	2	16	0	3
com.relive.routing.app.json		86%		n/a	4	23	2	26	4	23	0	3
com.relive.routing.app.constants		52%		n/a	4	5	4	6	4	5	3	4
com.relive.routing.app.models		90%		70%	4	14	1	21	1	9	0	2
com.relive.routing.app.custom.PopularRouteAlgorithm		100%		100%	0	11	0	31	0	10	0	3
com.relive.routing.app.custom.Weighting		100%		n/a	0	8	0	18	0	8	0	3
Total	1,081 of 2,895	62%	43 of 96	55%	93	247	226	627	63	199	8	38

Figure 50: Coverage results for the JUnit tests of the routing engine

```
yarn run mocha_test
yarn run v1.15.2
$ set NODE_ENV=TEST&&mocha ./build/test/route_builder/**/*.js


  Route Builder Tests
info: Create route called. user_id=1337, waypoints=[lat=52.011578, lon=4.357068, lat=51.011578, lon=4.357068], activityType=run, loggerName=routeBuilder, _c=0, _pid=master
info: GraphHopper response time duration=215, loggerName=routeBuilder, _c=1, _pid=master
    √ should be able to create a route (259ms)
info: Create route called. user_id=1337, waypoints=[lat=52.011578, lon=4.357068, lat=51.011578, lon=4.357068], activityType=run, loggerName=routeBuilder, _c=2, _pid=master
info: GraphHopper response time duration=100, loggerName=routeBuilder, _c=3, _pid=master
info: Update route called. waypoints=[lat=52.01148, lon=4.3572, lat=51.01139, lon=4.35707], activityType=run, route_id=1, user_id=1337, loggerName=routeBuilder, _c=4, _pid=master
info: GraphHopper response time duration=82, loggerName=routeBuilder, _c=5, _pid=master
    √ should, given a created route, be able to update the route (212ms)
info: Create route called. user_id=1337, waypoints=[lat=52.011578, lon=4.357068, lat=51.011578, lon=4.357068], activityType=run, loggerName=routeBuilder, _c=6, _pid=master
info: GraphHopper response time duration=83, loggerName=routeBuilder, _c=7, _pid=master
info: Delete route called. route_id=1, user_id=1337, loggerName=routeBuilder, _c=8, _pid=master
    √ should, given a created route, if it is deleted route should note be found (98ms)
info: Create route called. user_id=1337, waypoints=[lat=52.011578, lon=4.357068, lat=51.011578, lon=4.357068], activityType=run, loggerName=routeBuilder, _c=9, _pid=master
info: GraphHopper response time duration=78, loggerName=routeBuilder, _c=10, _pid=master
info: Get route called. route_id=1, user_id=1337, loggerName=routeBuilder, _c=11, _pid=master
    √ should, given a created route, it should be able to retrieve it (92ms)
info: Create route called. user_id=1337, waypoints=[lat=52.011578, lon=4.357068, lat=51.011578, lon=4.357068], activityType=run, loggerName=routeBuilder, _c=12, _pid=master
info: GraphHopper response time duration=84, loggerName=routeBuilder, _c=13, _pid=master
info: Create route called. user_id=1337, waypoints=[lat=52.011578, lon=4.357068, lat=51.011578, lon=4.357068], activityType=run, loggerName=routeBuilder, _c=14, _pid=master
info: GraphHopper response time duration=81, loggerName=routeBuilder, _c=15, _pid=master
info: Get routes called. user_id=1337, loggerName=routeBuilder, _c=16, _pid=master
    √ should, given multiple created routes, it should be able to retrieve all of them (218ms)
    √ should not allow us to create a route, if not authorized to do so


  6 passing (5s)
```

Figure 51: Results of the back-end tests.

E.1 Front-end Cucumber Scenario

```
Feature: Save Route
    As a user, I would like to save a created route.
    Scenario: Save Route
        Given Route length is greater than 0
        Then Wait for element: "route builder done button"
        And Click element: "route builder done button"
        Then Wait for element: "route builder list item"
        And Wait for element: "route builder edit route"
        And Click element: "route builder edit route"
        Then Wait for element: "route builder done button"
        And Click element: "route builder done button"
```

Listing 2: Front-end cucumber test scenario

F Survey Set-up & Results

F.1 Survey Questions

1. Did you have a chance to test the Relive route planner feature yet?
    - Yes
    - No
2. Why weren't you able to test this feature yet?
    - I didn't use the Relive app recently
    - I did not know this feature existed
    - I couldn't get the feature to work correctly
3. How easy was the feature to use?
    - Impossible to use
    - I found it hard to use
    - It worked OK
    - It was really easy to use
4. How often do you think you will use this feature in the future
    - Never
    - For some of my activities
    - For most of my activities
    - All of my activities
5. How could we improve this feature?
6. Can you tell us what went wrong?

F.2 Survey Results

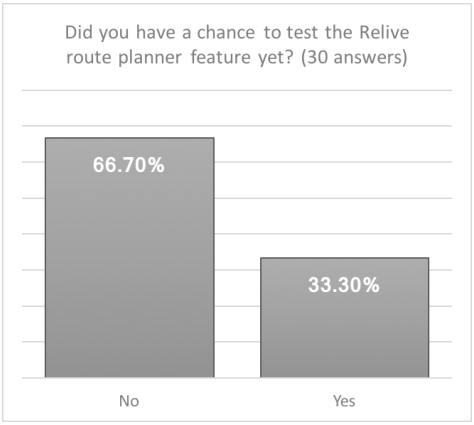


Figure 52: Survey Question 1

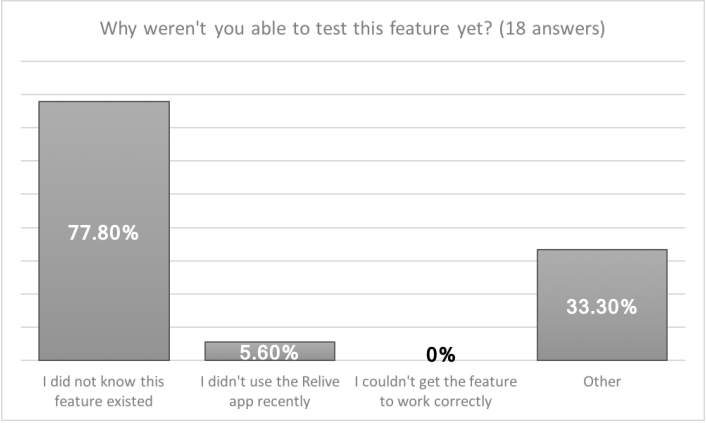


Figure 53: Survey Question 2

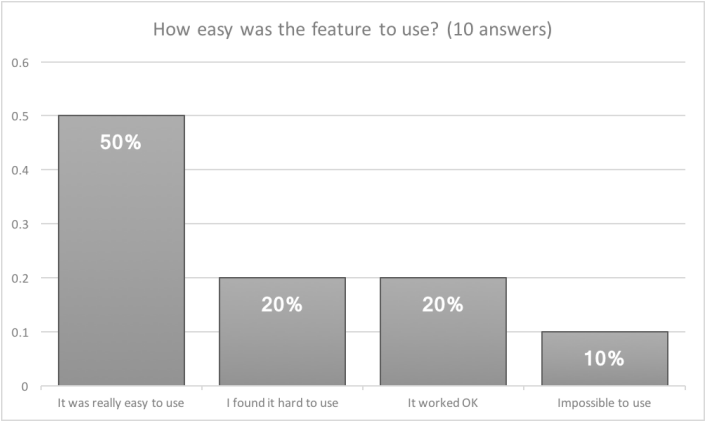


Figure 54: Survey Question 3

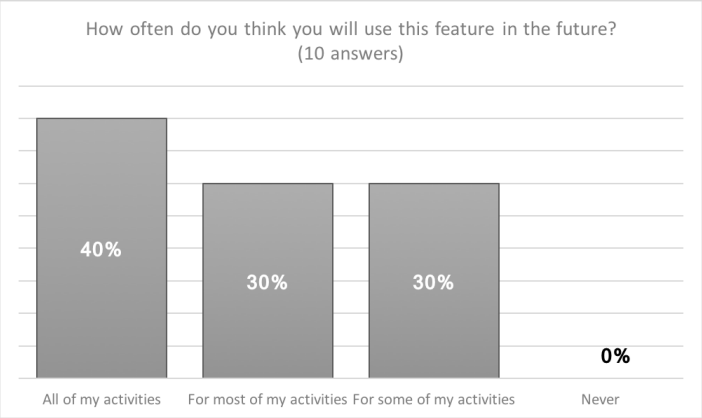


Figure 55: Survey Question 4

G Persona & User Stories

In this section several user stories will be discussed, taking several types of users into account. The types of users are as general as possible and are divided into: frequent but short activities, less frequent and normal activities, and rare but long activities. This division is based on the main types of the users. Different activity types are used for context, with relevant details of the users (training) goals. Some user stories are applicable to all users, but they do only appear once to prevent duplication.

G.1 Persona

The main personas of the project that will be used for the user stories are described here. The personas are based on a number of interviews (can be found in Appendix H) and our general understanding of the target audience of Relive.

Bob (cyclist) Bob is a young man aged 25 years. He cycles 3 times a week, sometimes recreationally and other times for commute. Each activity, Bob wants to take a different route. Bob likes to keep track of the statistics for each activity.

Alice (hiker) Alice hikes one time a week for a couple of hours. She likes to visit interesting places.

Tom (road-tripper) Tom goes on a road trip once a year with friends. He intends to plan a road trip that visits several cities, both popular and quiet spots.

Willem (runner) Willem runs a couple times a month. He likes to run a marathon and he is training for that.

G.2 User Stories

For each person described in the persona, a user story can be found in this section. Each story is based upon the estimated needs of the person. The user stories are focused on the route builder feature. The stories provide an insight into the incentive of the users to use the route builder. Each user uses the route builder in a different way and it is important that each use case is supported by the route builder.

Bob the cyclist (three times a week, 1-3 hours)

- As Bob, I want to plan a new route to cycle, in order to have a different route each time.
- As Bob, I want to be able to quickly create new routes, since I need to plan three new routes a week.
- As Bob, I want to see my previous activities in the map, since I don't remember all routes that I have cycled exactly, because I have cycled so much already.
- As Bob, I want to be able to change the route during the activity since it is possible that a route is closed or that I came up with a new destination to visit.
- As Bob, I want to share the routes I planned with my friends, so that we can enjoy the routes together, and to have the route available when we do the activity together.

- As Bob, I want to be able to use the application when I am not connected to the internet, so that when I am out in nature without a connection I can still plan what I want to do.
- As Bob, I want to see the routes of other activities, since it is easier to cycle without having to dodge runners or hikers.

Alice the hiker (one time a week, > 3 hours)

- As Alice, I want to spend time to plan a route since I only hike once a week and want the best route possible.
- As Alice, I want to see useful information (e.g., sightseeing points, road type, food/drink spots) regarding my planned route, so that I can make a more personalized and enjoyable route.
- As Alice, I want to see the route as detailed as possible (3D) in order to be able to realistically visualize the route and change it if needed.
- As Alice, I want to edit the routes of previous activities since some parts were particularly nice and I would like to hike those again.
- As Alice, I want to be able to see the paths others have travelled and see how popular certain paths are since I may hike in other countries where I am not familiar with the terrain.

Tom the road-tripper (once a year with friends)

- As Tom, I do not want to plan the entire trip on my own since my friends have their own preferences.
- As Tom, I want to be able to plan the entire trip (that takes several days) in advance, so that I can enjoy the trip without having to make a plan for the next day each time.
- As Tom, I want to be able to group several route plans since they are all for the same road trip.

Willem the Runner (couple of times a month, 0-2 hour)

- As Willem, I want to plan a route for exactly X kilometers since I follow a scheme to train myself for the next marathon of Amsterdam.
- As Willem, I want the same start and finish point and a circular route since I have just one house and I don't want to run the same route twice.
- As Willem, it would be nice to see the routes of other runners and cyclist since I want a road with the least amount of cyclists.

H Interviews

Interview set-up questions & criteria

1. What & why (+/-) do you use other apps other than Relive for any fitness activity?
2. What fitness activity do you do the most?
3. What scenarios occur that might trigger the need for route planning?
4. How can this scenario be solved optimally?
5. Gather the general profile of the person (age/fitness/gender).
6. Gather the expectations people have for the proposed application.
7. Gather the point of view.
8. Ask for any other pains/problems that the user encounters.

Interviewee Sample Explanation

All the interviewee's are employees at the Relive company. We used a primer question on whether they used Relive actively or not to filter out people who do not use the app actively. The sample we used is representative because most of the people working at Relive have sports such as hiking and cycling as a hobby, thus the information we've gathered come from people who have experience and would be included in the application's target audience as described in Section 2.2.2.

Interview #1

1. Cycles a few times per week
2. Does not really care about highlights (sightseeing) when making a route, mostly random points to make a route long enough
3. Creates a new route everytime when cycling long distances recreationally (to explore and prevent it from being boring)
4. Does not create a new route when commuting (not needed)
5. Would switch over to built-in Relive route planner (if better or equal)

Interview #2

1. Likes to take a different route each time
2. Would like to see heat-map functionality in the map to aid in making a better route (prevent seeing the same locations)
3. Would like to see 3D map in order to get a better overview of the terrain
4. Would like to see suggested sightseeing points in order to enhance the route
5. The weather when planning is not relevant (changes too much)

Interview #3

1. Mainly does cycling in a group
2. Currently uses a Garmin Tracker on the steering wheel of the bike to navigate throughout the trip and uses Relive to record the trip seperately.
3. When cycling in a group, most of the time only a single person navigates and keeps tracks of the road.
4. Mentions that other people mount their device on the wheel of the bicycle, however still uses Garmin Tracker to keep track of performance/power more accurately.
5. Mentions that when hiking, it is very easy to use and may be preferabe to use phone instead of a dedicated tracker/navigator.

Interview #4

1. Mainly cycles
2. Suggestion that it would be convenient to indicate one waypoint in the route planner to make it as easy, fast and simplified as possible. A circular route (or several routes) should be calculated around the point.

Interview #5

1. Mainly cycles on quiet roads
2. Given the activities of other users, it is best to follow roads that have previous activities of users to avoid bad roads
3. Given the activities of other users, the preference would not be the road with the most activities of other users (of any activity type), since that can possibly indicate that the road is too busy.

I Software Improvement Group (SIG)

First SIG Feedback

De code van het systeem scoort 3.9 sterren op ons onderhoudbaarheidsmodel, wat betekent dat de code marktgemiddeld onderhoudbaar is. We zien Unit Sizevanwege de lagere deelscore als mogelijke verbeterpunten.

Bij Unit Size wordt er gekeken naar het percentage code dat bovengemiddeld lang is. Dit kan verschillende redenen hebben, maar de meest voorkomende is dat een methode te veel functionaliteit bevat. Vaak was de methode oorspronkelijk kleiner, maar is deze in de loop van tijd steeds verder uitgebreid. De aanwezigheid van commentaar die stukken code van elkaar scheiden is meestal een indicator dat de methode meerdere verantwoordelijkheden bevat. Het opsplitsen van dit soort methodes zorgt er voor dat elke methode een duidelijke en specifieke functionele scope heeft. Daarnaast wordt de functionaliteit op deze manier vanzelf gedocumenteerd via methodenamen.

Voorbeelden in jullie project:

- BenchmarkScripts.main(String[])
- PolylineDecoder.decodePoly(String)
- HibernateUtil.getSessionFactory()

Als laatste nog de opmerking dat er geen (unit)test-code is gevonden in de code-upload. Het is sterk aan te raden om in ieder geval voor de belangrijkste delen van de functionaliteit automatische tests gedefinieerd te hebben om ervoor te zorgen dat eventuele aanpassingen niet voor ongewenst gedrag zorgen. Op lange termijn maakt de aanwezigheid van unit tests je code ook flexibeler, omdat aanpassingen kunnen worden doorgevoerd zonder de stabiliteit in gevaar te brengen.

Over het algemeen is er dus nog wat verbetering mogelijk, hopelijk lukt het om dit tijdens de rest van de ontwikkelfase te realiseren.

Second SIG Feedback

In de tweede upload zien we dat het codevolume is gegroeid, terwijl de score voor onderhoudbaarheid is gestegen.

We zien dat de verbeterpunten uit de feedback op de eerste upload zijn aangepast, en op deze gebieden is dan ook een verbetering in de deelscores te zien. We zien ook dat de nieuwe code ook op andere gebieden beter onderhoudbaar is geworden in vergelijking met de reeds bestaande code.

Het is goed om te zien dat er naast nieuwe productiecode ook nieuwe testcode is geschreven.

Uit deze observaties kunnen we concluderen dat de aanbevelingen van de vorige evaluatie grotendeels zijn meegenomen in het ontwikkeltraject.

J Project Description

3D Route Builder

Key areas: Software Engineering + Data Visualisation

Based on our dataset of nearly [redacted] GPS-tracked activities, can we build the perfect route builder for any outdoor enthusiast?

Routing engines are a complex Software Engineering problem. Perhaps we can use an engine such as Valhalla (https://github.com/valhalla)? Valhalla is an open source routing engine and accompanying libraries for use with OpenStreetMap data. Valhalla also includes tools like time+distance matrix computation, isochrones, elevation sampling, map matching and tour optimization (Travelling Salesman).

Suggested technologies:

- Open streetmap
- Valhalla
- Frontend web application using React
- Client-side map engine such as Leaflet, or the 3D javascript map engine such as used in our Explorer tool (https://www.relive.cc/view/447820220/explore)

Inspiration:

- https://www.komoot.com/plan

About Relive

Join the 3D Outdoor Platform for Millions of Users

Built by ex-TU Delft students - used all around the world! In just two years, over a million cyclists, runners, hikers and skiers have joined Relive to share their passion & adventures - making Relive one of the top 30k websites world-wide and fastest growing Dutch startups.

Relive turns outdoor adventures into short video animations that people love to share with family and friends. Relive is all about the total experience of your trip. No focus on performance metrics and competition. Let's leave that to traditional sports trackers.

Check out a Relive of our recent team-trip to Tenerife:

https://www.facebook.com/relivecc/videos/315334475537897

Early adopters include Laurens ten Dam, Lance Armstrong and Instagram founder Kevin Systrom!

On both the technical and business side; Relive offers many challenges in different areas (ranging from Data Science, Mobile + Web Development, Infrastructure Engineering, 3D Computer Graphics). Every day we process GPS data from 200.000+ activities, and turn these into a video using our dynamically scaling infrastructure.

Your BSc. project at Relive

Look for Relive in the bepsys directory for different projects to give you an idea what you could be working on at Relive. Of course there's room for your own suggestions; drop by our office next to Rotterdam CS and let's find a fit with your interests!

K Info Sheet

On the next page the info sheet for this project can be found.

Info sheet

Project title 3D Route Builder
Client Relive B.V.
Final presentation July 2nd, 2019

Description

The last four years, the Relive sport app experienced rapid growth and therefore the company aims to increase the retention rate of the current 4 million users. Relive gained a healthy reputation due to a feature that transforms an outdoor sporting activity to a memorable 3D movie. The goal of the project is to develop a new feature that increases the retention rate of the app.

This new feature entails a route builder that allows users to plan routes for their hiking, biking and running activities. Other route planner apps do exist; however, it is beneficial for Relive to keep users inside the app to improve retention rates. During development, agile processes are used to quickly adapt to requirement changes that form out of new insights. The research phase resulted in several tools and languages that are used throughout the project.

The project tackles the calculation of a route based on a given set of points, to allow for user interaction with a map, and to render the routing result. Throughout the project, an experiment is performed with 22.000 users to test the product and provide feedback in the form of surveys and event tracking. The final product is a feature inside the Relive app that allows users to plan a route in several maps, including 3D. Additionally, a proof of concept for a routing engine based on the most popular routes has been successfully created.

Members of the project team

Name: Rowdy Chotkan
Interests: Computer Science, Front-end Development, User Experience, and Testing
Role and Contribution: Front-end Development and Testing, User Tests, and User Experience

Name: Mika Kuijpers
Interests: Web (full stack) development, creating user oriented solutions, Algorithm design
Role and Contribution: Full stack development and testing, User Experience

Name: Paul van der Laan.
Interests: Back-end development, algorithms, big and impactful applications and user contact
Contribution and Role: Developer and tester (mainly back-end), (customized) routing algorithm implementation

Name: Brian Planje
Interests: Full Stack Web Development, UI/UX Design
Contribution and role: Developer and tester (mainly back-end), (customized) routing algorithm implementation

Client

Relive, Yousef El-Dardiry (Co-Founder), Frikkie Snyman (Supervisor)

Coach

Sander van den Oever, Department of Computer Science

Contact

Paul van der Laan paulvanderlaan@live.nl
Rowdy Chotkan rowdy_chotkan@msn.com

The final report for this project can be found at: http://repository.tudelft.nl

L Asana

On the following pages follow the tasks completed during the project.

Id	Name	Created	Done
0	Create a summary of the app	23-Apr	24-Apr
1	Create a summary of app users and how they use the app	23-Apr	24-Apr
2	Create a summary of what other apps are used to achieve the same route building experience	23-Apr	29-Apr
3	Familiarize with React Native	23-Apr	29-Apr
4	Try out an OSX VM	23-Apr	01-May
5	Create a MoSCoW requirement document.	23-Apr	24-Apr
6	Find a solution for the MacBook issue.	23-Apr	25-Apr
7	Perform in-house interviews to gather information.	23-Apr	25-Apr
8	Create user stories & a persona.	23-Apr	24-Apr
9	Meet with Sander (TU Delft coach)	23-Apr	25-Apr
10	Research Report	24-Apr	
11	Find useful libraries	24-Apr	29-Apr
12	Research Report - Introduction	25-Apr	26-Apr
13	Research Report - Problem Statement	25-Apr	26-Apr
14	Research Report - Development Tools	25-Apr	26-Apr
15	Research Report - Requirements	25-Apr	26-Apr
16	Remove duplication from report	26-Apr	29-Apr
17	Finish routing libraries	26-Apr	29-Apr
18	Read through report	26-Apr	26-Apr
19	Fix typos and formatting issues	26-Apr	26-Apr
20	Setup the Relive app locally	29-Apr	29-Apr
21	Choose a first routing API	29-Apr	30-Apr
22	Implement an adapter pattern for Routing API's	29-Apr	30-Apr
23	Set-up connection between the application and the routing API	29-Apr	30-Apr
24	Add a tab to the Relive app	29-Apr	30-Apr
25	Show a map inside the new tab	29-Apr	30-Apr
26	Ability to place markers on the map	29-Apr	30-Apr
27	Draw a route between markers	29-Apr	30-Apr
28	Implement feedback from Coach	30-Apr	01-May
29	Extend GraphHopper connection functionality	30-Apr	01-May
30	Start with OSRM or Valhalla connections	30-Apr	01-May
31	Update research report based on feedback	30-Apr	30-Apr
32	Store whole route response instead of route markers	01-May	02-May
33	Show route stats using route response	01-May	02-May
34	Add waypoint numbers to indicate order of the waypoints	01-May	02-May
35	Contact about a possible server and API options (in particular GraphHopper, very good API, but paid)	01-May	13-May
36	Display route on tracker screen	01-May	03-May
37	Look into deliverables TU Delft	02-May	02-May
38	Implement route overview screen	02-May	09-May
39	Finalize report	02-May	03-May
40	Update route between new waypoints	03-May	27-May
41	Refactor RouteBuilder	03-May	06-May
42	Fix POI interaction	03-May	13-May
43	Change marker colour	03-May	03-May

44	Polyline fix	03-May	03-May
45	Add markers on route between markers	03-May	27-May
46	Snap markers to road after receiving route from API	03-May	08-May
47	Close route to start point	03-May	08-May
48	Provide better user feedback on bad request	03-May	22-May
49	To delete marker, drag to recycle bin	03-May	13-May
50	Improve route stats bar (readability and hh:mm)	03-May	10-May
51	Spread info on top bar out, instead of centering everything on the top bar	03-May	10-May
52	Add button for user to press to import route into tracker	06-May	10-May
53	Save route to database	06-May	10-May
54	Create endpoint save the route	06-May	10-May
55	Create endpoint to get route	06-May	10-May
56	Create endpoint to delete route	06-May	10-May
57	Set up skyhawk-website	06-May	07-May
58	Integrate existing route api into skyhawk-website	06-May	10-May
59	Get routes from server to local device for offline selection	06-May	22-May
60	Displaying route while offline	06-May	10-May
61	Inform about possible user feedback possibilities for the test 'release'	08-May	13-May
62	Change color begin and end marker	09-May	22-May
63	Delete route (front-end)	09-May	13-May
64	Clear route from tracker	09-May	17-May
65	Use navigation parameters instead of stores while navigating	10-May	
66	Edit route from route overview screen	13-May	15-May
67	Goto current location in mapview	13-May	17-May
68	Map type button in mapview	13-May	14-May
69	onActivityChanged doesn't check for waypoints > 1, so throws error	13-May	13-May
70	Update route on marker drag	13-May	
71	Animate route drawing	13-May	
72	On route select, snap to route	13-May	17-May
73	Reduce size select route button in tracker screen	13-May	13-May
74	Update Research Report	13-May	
75	Update Product Planning	13-May	
76	backend: add checks to not try to calculate a route with 1 waypoint	13-May	16-May
77	Add search bar	13-May	
78	Get a paid graphhopper api key	13-May	15-May
79	Demo Goals	13-May	
80	GPX export from route	13-May	
81	Event tracking	13-May	15-May
82	Set up survey	13-May	31-May
83	Make direction of route clear in tracker	13-May	
84	Fix Android onscreen back button	13-May	15-May
85	Optimize call order	13-May	
86	Website feedback	14-May	20-May
87	Look into a fix for different mapTypes on iOS (3D map)	14-May	27-May
88	Refactor modals, ListModalMenu can be used for almost all modals	15-May	21-May
89	Add done button in route builder	15-May	21-May

90	The route should clear when done recording or pressing cancel from the record screen	15-May	16-May
91	Update code formatting	16-May	16-May
92	Add text to indicate empty route list on `Select Route` screen	16-May	16-May
93	Catch error when GPS is disabled (RouteBuilderMap)	17-May	27-May
94	Make introduction text non sticky	17-May	17-May
95	Handle iPhone back swipe event	17-May	17-May
96	Make the card button on create/select screen of planned routes smaller	17-May	20-May
97	Refactor inline styling to stylesheets	17-May	27-May
98	Add current location button	17-May	27-May
99	Finish saving/editing route names	20-May	20-May
100	Use dictionaries	20-May	22-May
101	Get access to the routes db	20-May	27-May
102	Set up graphhopper locally	20-May	27-May
103	Deploy our own grapphopper	20-May	27-May
104	Showcase our own graphhopper	20-May	11-Jun
105	Display activity type in Route Overview Screen	20-May	23-May
106	Update route in tracker	20-May	21-May
107	Add select route button to tracker recording screen	20-May	24-May
108	Activity type get's changed to default on route edit	21-May	27-May
109	Remove time estimation	21-May	22-May
110	Add user profile units	21-May	21-May
111	Link backend microprocessor cache to end-to-end cloud mainframe	21-May	21-May
112	Fix empty route list background position	21-May	03-Jun
113	Back-end should return route with activity type	21-May	22-May
114	No internet screen in route editor	21-May	26-May
115	Disable 'your location' on iOS if you click blue dot in map	21-May	
116	Route calculation optimisation	22-May	27-May
117	Markers between markers	22-May	
118	Returning to route list and quickly tapping on map adds a waypoint	23-May	
119	Move trash bin to separate button	24-May	03-Jun
120	Fix record activity types	26-May	27-May
121	Backlog	27-May	
122	Make route shareable	27-May	17-Jun
123	ETA from user's activities	27-May	
124	3D map	27-May	04-Jun
125	Frontend: see if we can use existing heatmap feature to show user where they have been previously	27-May	04-Jun
126	Backend: use routes database to get most popular sections (heatmap)	27-May	17-Jun
127	Heatmaps	27-May	11-Jun
128	Get routes is called for users not in the experiment	27-May	30-May
129	Off road support	27-May	
130	Get dashboard up	27-May	30-May
131	Move average speed PR to back-end #1425	29-May	
132	Release 2 - Week 7	31-May	11-Jun
133	UI meeting points Sander	31-May	11-Jun
134	Make route name editing more clear	31-May	03-Jun

135	Submit to SIG	31-May	31-May
136	Improve first usage experience	31-May	03-Jun
137	Create interactive feature intro	03-Jun	05-Jun
138	Move tab button to tracker screen	03-Jun	05-Jun
139	Make route selectable during record	03-Jun	03-Jun
140	Send more surveys	03-Jun	05-Jun
141	Add some beta testers (opt-in)	03-Jun	11-Jun
142	See if more users go from start to create	03-Jun	
143	See if there is an increase from create to select	03-Jun	
144	Create a start button from the route list/mapview	03-Jun	05-Jun
145	Invite some users for a Skype interview	03-Jun	
146	Get some more specific usage feedback from users	03-Jun	11-Jun
147	Increase experiment intake to current users	03-Jun	
148	Arrange meeting with Sander	05-Jun	07-Jun
149	Enable route tab for all users	06-Jun	07-Jun
150	Route is still selected after deleting the route	06-Jun	07-Jun
151	Draw route on top of heatmap	06-Jun	06-Jun
152	Back button improvement (see description)	06-Jun	06-Jun
153	Bottom button reuse (see description)	06-Jun	06-Jun
154	Heatmaps with 3D map not working nicely	06-Jun	
155	FIgure out why Yousef's app crashed	06-Jun	17-Jun
156	While tracking, route get deselected if you leave app for some time	06-Jun	07-Jun
157	Add ? button to show instructions again	06-Jun	06-Jun
158	Optimisation: remove inline functions	07-Jun	
159	Improve RouteBuilderOnBoardingModal (not using StartupModal)	11-Jun	
160	SIG feedback implementation	11-Jun	
161	Implement UI/UX changes as discussed with Sander	11-Jun	
162	Import GPX files that creates a route	11-Jun	
163	Final Report	11-Jun	
164	Submit to SIG	11-Jun	
165	Fix route distance	17-Jun	
166	Finish route sharing	17-Jun	