Live Tracking Software for Outdoor Events using GPS Trackers on Athletes

Clemens Stary

Live Tracking Software for Outdoor Events using GPS Trackers on Athletes

Clemens Stary B.Sc.

Master's Thesis

to achieve the university degree of

Diplom-Ingenieur

Master's Degree Programme: Computer Science

submitted to

Graz University of Technology

Supervisor

Univ.-Prof. Dipl.-Ing. Dr.techn. Alexander Felfernig
Institute of Software Technology (IST)

Graz, 10 Jul 2019

Abstract

Live tracking sporting events is becoming more popular and nowadays is used for many events. Numerous different tracking solutions (available to buy or rent) are used to enhance the experience for spectators, moderators, and athletes.

The created live tracking application consists of a client and a server. The client is a web application optimized for desktops and mobile devices. A page presents the current and past events and supports filtering them. The live tracking page visualizes the athletes of a live event and displays multiple statistics such as average speed, estimated arrival or pace. To track an athlete's performance, charts for functions such as speed or pace are also available. Besides the athlete, the live tracking page can also display a racetrack that may consist of multiple legs and points of interest which can trigger notifications. Since GPS devices are not always accurate, the software provides multiple mapping algorithms to align the athlete track to roads or the race track. Furthermore, all pages support multilingual input.

The developed application uses a Java server which gathers data about the locations of the GPS devices from an external database. It produces the data structures used by the client that include charts, statistics, and location data or mapping the GPS data. The client can access the data by using the implemented representational state transfer interface. The main contributions of this thesis are the various mapping algorithms which make it possible to align the athlete's track to roads or on the racetrack. Furthermore, the application offers many different statistical evaluations for athletes which are calculated in real time. Besides the statistical evaluations, the performance of an athlete can also be visualized in a chart using the calculated real-time data. Another feature allows others to access the calculated statistics in order to create their own analysis. The user interface of the web has a special design to make it usable for any device, for example, desktops, tables, and mobile phones.

Contents

List of Figures

List of Tables

List of Listings

Acknowledgements

I would like to thank my advisor Alexander Felfernig for offering me the possibility to create this application and for his support and feedback.

I would like to thank Christoph Uran from the Fachhochschule Kärnten for sharing his knowledge about the live tracking domain and helping me find solutions for my tasks. I would also like to thank the Fachhochschule Kärnten for allowing me to using their tracking system which made the creation of this application possible.

I would like to thank Meredith Brown for proofreading this thesis.

Clemens Stary
Graz, Austria, 10 Jul 2019

Credits

I would like to thank the following individuals and organizations for permission to use their material:

- This thesis was written using Keith Andrews' skeletonthesis .

Chapter 1

Introduction and Motivation

In recent years, the live tracking domain has gained more and more popularity due to the technical progress and easier access to live tracking devices. A live tracking system consists of three parts; firstly, a tracking unit which automatically sends locations data to the server (called a global positioning system (GPS) tracker), the second is a system that provides a communication interface for GPS trackers to transmit the data and process them to accomplish several tasks, and the third part is a system that presents the data to the users,for instance, on a web site or a mobile application.

In the past, GPS trackers have been bigger and not as practical to transport, and live tracking systems were only used by the military. At the time, the public usage of GPS tracking was not as simple as it is now as there was a lack of GPS satellites. Fleet tracking was one of the first public scopes used for live tracking. Companies could track all of their vehicles or boats and used the position data to improve route planning, fuel consumption or to predict the arrival time [Pham et al. 2013; Darekar et al. 2012].

The technical progress has improved the accuracy of GPS devices and has made them much smaller. In addition, the expansion of mobile internet has made data transmission much cheaper. These improvements have made it possible to use live tracking on a wider scope, so that it can now be used for live tracking sport events, to name an example. Nowadays, live tracking is very popular and used by a majority of athletes to track performance and to improve their skills [Ahtinen et al. 2008]. Today, live tracking systems are used in several sports, such as soccer, football, tennis, marathons, or other outdoor events. The goal of live tracking is to provide real-time information about athletes, which are useful for spectators, moderators, coaches, and the athletes themselves.

The goal of this thesis is to create a tracking system for outdoor sport events such as running, cycling, marathons, duathlons, or Ironman triathlons in cooperation with the Fachhochschule Kärnten.
The goal is to create a complete tracking system consisting of multiple parts. In cooperation with the FH Kärnten it was possible to use an already existing interface to collect the location data of GPS devices. The collected data is stored in predefined database structures. This has the advantage that, compared to other solutions, it does not limit the use to specific GPS devices. The interface supports several protocols for GPS trackers and mobile phones. This makes it possible to use different kinds of devices depending on the activity, such as more compact and lighter trackers for running events or dust and waterproof trackers for spartan races or duathlons.
The task of the tracking system is to access and process the data from the database and present them to the user. The tasks of the server include using the athlete's data and calculating real-time statistics to evaluate their performance. To show their progress, the server creates a data structure for external clients which can be used as input for a graph visualization tool, such as Google Charts[1]. Another task is to use different mapping methods to show the correct position of athletes even if the GPS tracker is not very

[1]Google, *Google Charts*. `https://developers.google.com/chart/` (visited on 20/03/2019).

accurate. These three core features should be flexible and easy enough for other developers to expand. In addition to this, the goal is to provide a uniform interface for external clients to access the produced data via the internet. This makes it possible to use the interface and create easier applications for other operating systems.
In this case the client is a web application that displays the data to the user. The web application consists of two important parts: the presentation and filtering of events, and the live tracking page, which visualizes the athletes on a map. Both of these pages can be accessed by any web browser, whether on mobile phones or desktops. Furthermore, the web application uses the provided server's interface to access the data and visualize it for the users. Points of interest can be placed along the racetrack to create notifications when athletes are near them, which can be modified accordingly by event managers on the web application to upload or change them.

Compared to other live tracking systems, this combination of implemented features is unique. Alternative systems habitually focus on a better live tracking visualization but lack statistical evaluation or vise versa. The major contributions of this thesis are the various mapping algorithms which can align the athletes' track to roads or on the racetrack. In comparison to other solutions, the mapping of athletes is a unique feature that is not provided by other applications. Furthermore, the varieties of statistical evaluations of athletes are one of a kind. Other solutions neither provide the number of real-time charts, nor do they state statistical evaluations. Additionally, the data is accessible for others to create their own analysis. Besides those features, the user interface of the web application can be used by any device, for example, desktops, tables, and mobile phones.

The remainder of this thesis is organized as follows: Chapter 2 evaluates current used live tracking solutions their technical implementation and other domains that uses athlete tracking. Since the live tracking application is created in cooperation with the Fachhochschule Kärnten, an important part is to create requirements and research usable software designs. Chapter 3 describes the process to create requirements and choosing the right software system. Chapter 4 explains the used technologies and the implemented software components including the choices of design made in the creation process. The uniform interface called representational state transfer (REST) is introduced in Chapter 5. This chapter covers the terminology and important design guidelines to create REST interfaces. Chapter 6 gives an overview of the created web application used by the client and discusses the components of the used design patterns. The live tracking application have been tested during live events and the results are presented in Chapter 7. Finally, Chapter 8 discusses features which are still in progress and ideas for future development.

Chapter 2

Related Work

2.1 Live Tracking Systems

Athlete tracking is becoming more and more relevant for athletes, spectators, and moderators. The fast technological progress makes it possible to use smaller tracking devices and send the location data in real time to the server. Systems process the received coordinates and represent them to the users via different devices, such as smart phone applications, web sites or desktop applications.

Available systems on the market provide various kinds of solutions. Some companies have offered to buy or rent a full tracking package including trackers and access to the software application. Using these packages is often more costly and limited to the provided features of the seller. Other providers allow one to buy a compatible tracking device from the store and use their tracking app for free. These tracking devices are often global positioning system (GPS) trackers and use the mobile internet to send the data to the server, requiring a SIM card to access the mobile web. Nowadays, more and more athletes use their own tracking devices during an event to evaluate their performance, such as smart phones. Tracking apps for smart phone makes it possible to use the device as a GPS tracker. These tracking apps present the recorded track and statistical evaluations within the app. In most cases the data is also available on online platforms, where other users can access them. Such free tracking solutions often limit the features and result in extra charges for premium functionalities, such as advanced statistical evaluations or live tracking options. There are many tracking systems with different advantages and disadvantages which have influenced the outcome of this project.

2.1.1 Runtastic Running App & Run Tracker

Runtastic Running App & Run Tracker is an application available for Android and iOS, developed from the Austrian company Runtastic[1]. The Runtastic tracking app turns an Android or iOS device into a GPS tracker. To use the application the smart phone must have a GPS module and active internet connection.

2.1.1.1 Create and Select a Route

Before starting a live event, it is possible to select a race track on which the user plans to run. A route can either be created by other users on the Runtastic platform, or the runners can make their own using the online creation tool. Figure 2.1 shows the web site where users can create a custom track. By clicking on the map, the tool creates reference points which connect the path of the race track. On the left side of the user interface there is a control window which offers the function to remove reference points, close the track, or align the created track to the street. By clicking the "Save" button placed on the bottom

[1]Runtastic, *Running, Cycling & Fitness GPS Tracker*. `https://www.runtastic.com/` (visited on 20/03/2019).

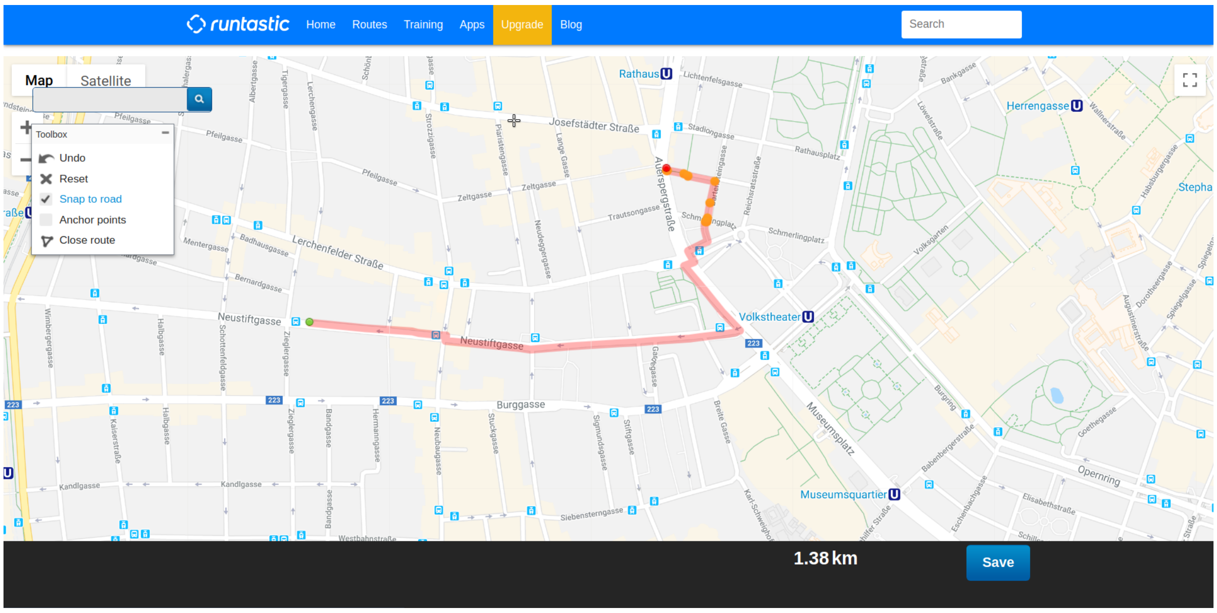


Figure 2.1: The Runtastic web site to create a running track. [Screenshot taken by the author of this thesis from Runtastic.]

right side, the route can later be selected again before starting an event on the smart phone application, as shown in Figure 2.2. The route selection is optional.

2.1.1.2 Start Event and Live Tracking

After selecting the route, the live tracking starts by clicking on the green "Start Event" button, placed on the bottom of the screen, seen in Figure 2.3. Starting the live tracking allows other users to follow the live event on the Runtastic web site.

Figure 2.4 shows the user interface of the live tracking web site. The name of the athlete is shown on the top of the page under the navigation bar. Live statistics of the athlete are placed above the tracking map. The athlete's position on the map as well the statistics are updated in real time. Available statistics include covered distance, duration, pace, burned calories, elevation gain and heart rate (if the user connects their device with a smart phone). A statistical analysis for every covered kilometer is placed on the right side of the map. It shows the pace and averaged speed for each milestone and also the duration it took to cover. In addition to this, elevation gain and loss is shown for every milestone.

The map, which shows the live tracking of the athlete, is placed in the center of the web page. The athlete's track is represented by a blue line. The current position is the marker with a picture of the athlete. Yellow markers along the track indicate the milestones. Clicking on the "Center map" button placed on the bottom right of the map changes the perspective to display the whole track. Enabling the "Follow Mode", placed on the bottom left of the map, will always center the map to the athlete's current position. The map can be changed between two web mapping services Google Maps[2] and OpenStreetMap[3] using the control field on the top. Selecting a route before starting the event will show it on the map before initiating the course, it will not however be visible during the run. It is used to compare the performance with other athletes who run the same route. Each live event only contains individual athletes and cannot contain multiple participants' visibility on the map.

[2]Google, *Google Maps*. `https://maps.google.com/` (visited on 20/03/2019).
[3]OpenStreetMap. `https://www.openstreetmap.org` (visited on 20/03/2019).

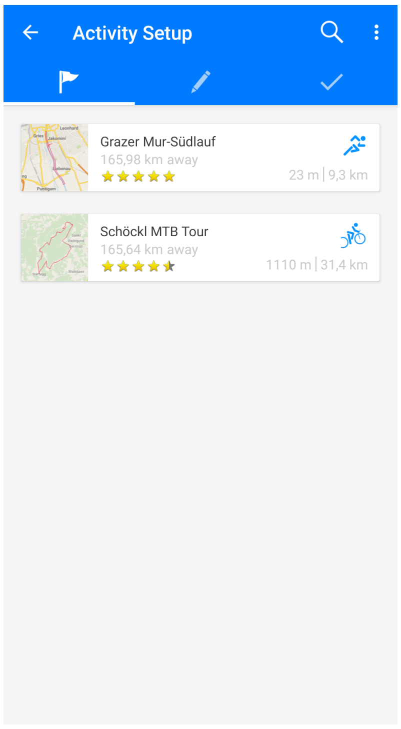


Figure 2.2: The route selection page on the Android application. [Screenshot taken by the author of this thesis from Runtastic.]

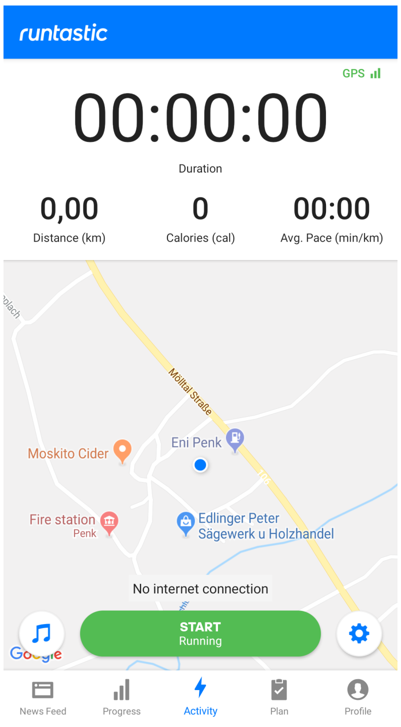
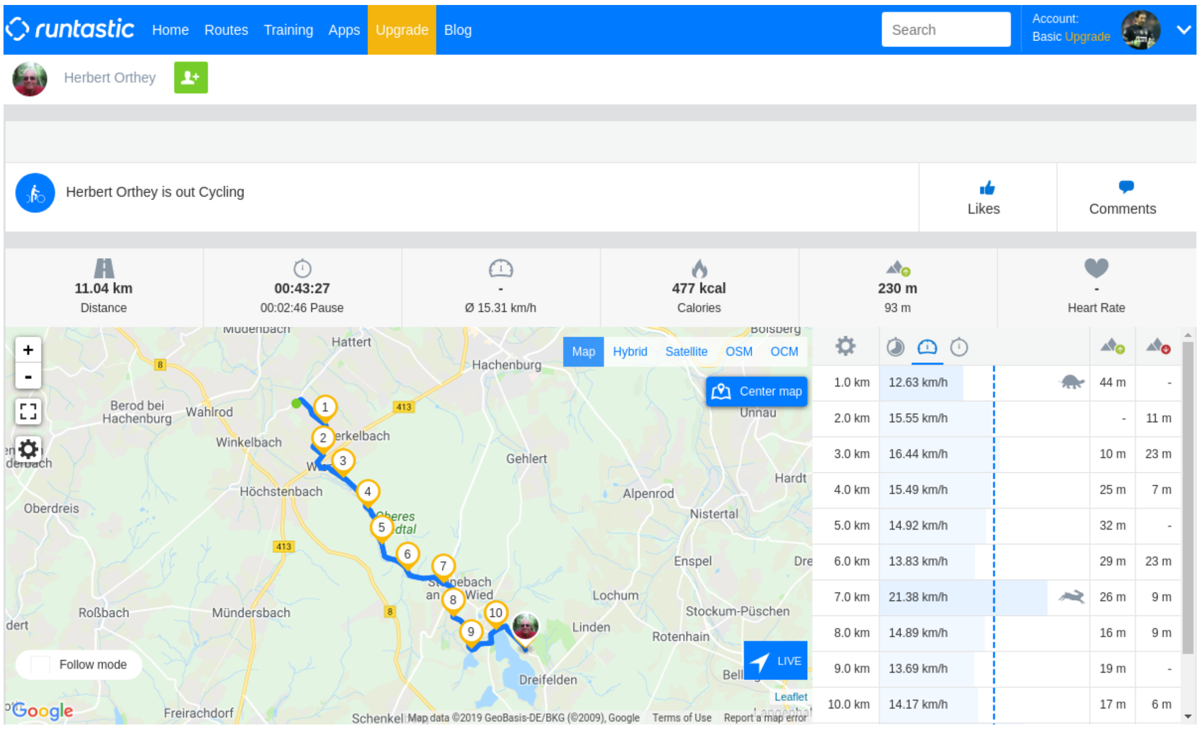


Figure 2.3: The page to start the live tracking of the Android application. Pressing the "Start" button starts the live tracking. [Screenshot taken by the author of this thesis from Runtastic.]



Herbert Orthey is out Cycling

11.04 km	00:43:27	-	477 kcal	230 m	-
Distance	00:02:46 Pause	Ø 15.31 km/h	Calories	93 m	Heart Rate

1.0 km	12.63 km/h	44 m	-
2.0 km	15.55 km/h	-	11 m
3.0 km	16.44 km/h	10 m	23 m
4.0 km	15.49 km/h	25 m	7 m
5.0 km	14.92 km/h	32 m	-
6.0 km	13.83 km/h	29 m	23 m
7.0 km	21.38 km/h	26 m	9 m
8.0 km	14.89 km/h	16 m	9 m
9.0 km	13.69 km/h	19 m	-
10.0 km	14.17 km/h	17 m	6 m

Figure 2.4: The Runtastic web site to view the live tracking. [Screenshot taken by the author of this thesis from Runtastic.]

2.1.1.3 After Race Evaluation

The user has to manually stop their live tracking on the smart phone by pressing the "Finish" button on the bottom right, as shown in Figure 2.5. After stopping the live tracking, the applications shows a resume of the event on the mobile phone, seen in Figure 2.6. Figure 2.7 shows the resume which includes the same statistics as mentioned in Section 2.1.1.2. Additionally, a chart showing the speed, pace, and altitude profile is generated. The chart is accessible via the web page and the mobile application, as shown in Figure 2.8 and Figure 2.9.

2.1.1.4 Conclusion

Runtastic is an easy to use tracking system. All the user needs to do is install the tracking application on their smartphone and create an account. The disadvantage to using this option is that the live tracking is not accessible unless the user shares a link and there is no list that displays all active users. Another con is that the application cannot display multiple users at once. Due to that reason, Runtastic is only a practical for single user tracking. The web page and the map is static, and does not allow any customization such as creating points of interest.

2.1.2 Tractalis

Tractalis[4] is a company which offers a live tracking solution for sport events. The tracking solution can be rented and includes GPS trackers for athletes and access to event management sites. The web site does not provide any information about the cost for renting the live tracking service. Every athlete assigned to a GPS device stores information such as their name, age, gender, and categories of athletes. This information is used by the live tracking view to filter athletes.

[4]Tractalis, *Tracking system for your event*. `https://www.tractalis.com/` (visited on 20/03/2019).

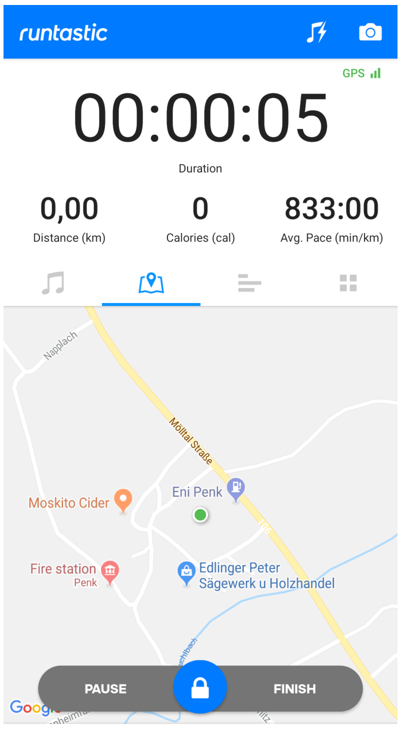
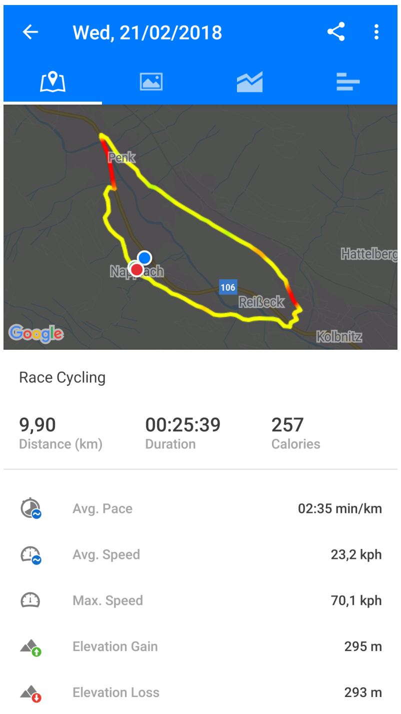


Figure 2.5: The page to stop the live tracking of the Android application. Pressing the "Stop" button stops the live tracking. [Screenshot taken by the author of this thesis from Runtastic.]

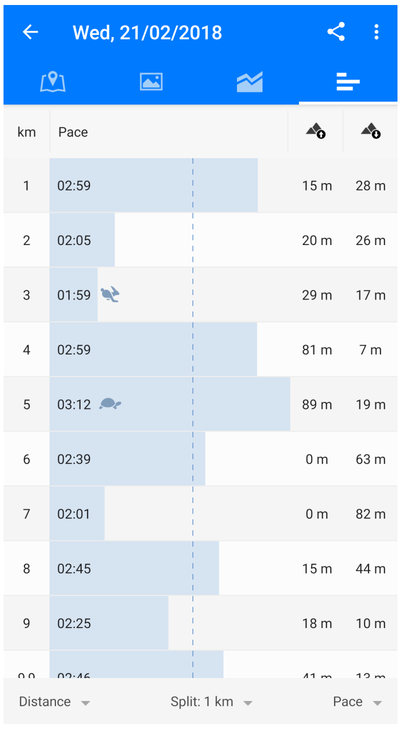


Race Cycling

9,90	00:25:39	257
Distance (km)	Duration	Calories

Avg. Pace	02:35 min/km
Avg. Speed	23,2 kph
Max. Speed	70,1 kph
Elevation Gain	295 m
Elevation Loss	293 m

Figure 2.6: The page of the web application shows a statistical evaluation after the live event. [Screenshot taken by the author of this thesis from Runtastic.]



km	Pace		
1	02:59	15 m	28 m
2	02:05	20 m	26 m
3	01:59	29 m	17 m
4	02:59	81 m	7 m
5	03:12	89 m	19 m
6	02:39	0 m	63 m
7	02:01	0 m	82 m
8	02:45	15 m	44 m
9	02:25	18 m	10 m
9.9	02:46	41 m	12 m

Distance Split: 1 km Pace

Figure 2.7: The page of the Android application shows a statistical evaluation for every mile stone after the live event. [Screenshot taken by the author of this thesis from Runtastic.]

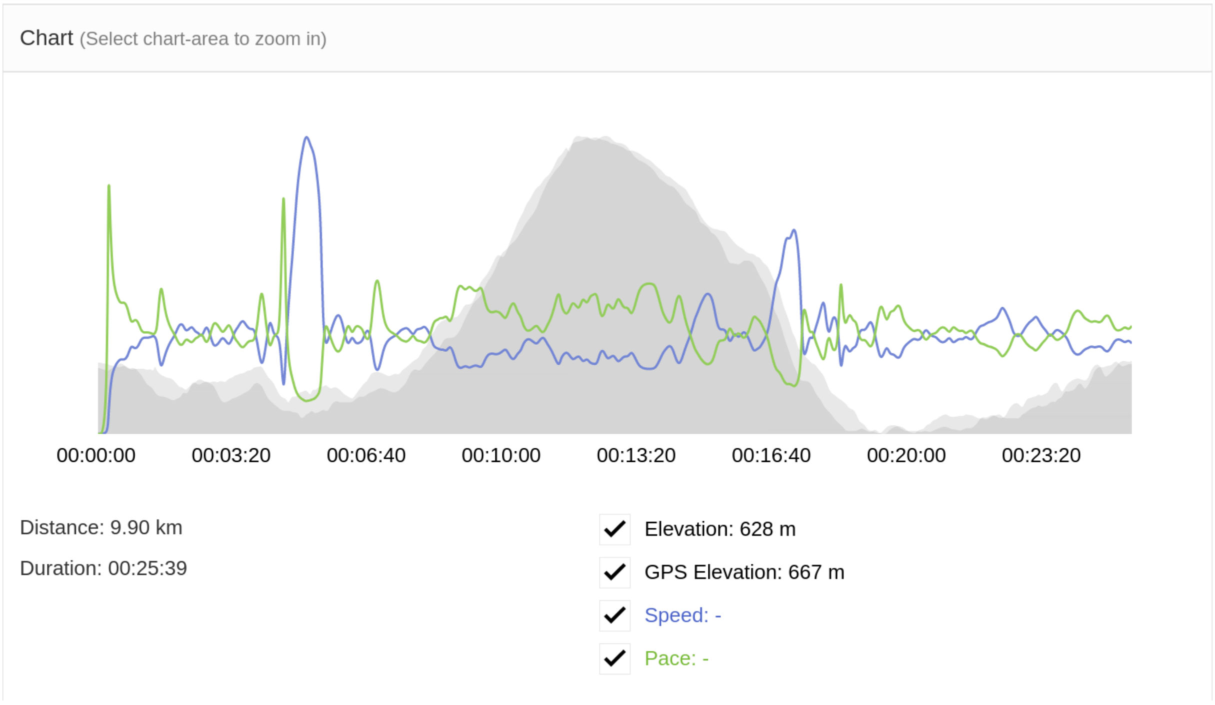


Chart (Select chart-area to zoom in)

Distance: 9.90 km

Duration: 00:25:39

✓ Elevation: 628 m

✓ GPS Elevation: 667 m

✓ Speed: -

✓ Pace: -

Figure 2.8: The chart which is displayed after the live tracking on the Runtastic web site. [Screenshot taken by the author of this thesis from Runtastic.]

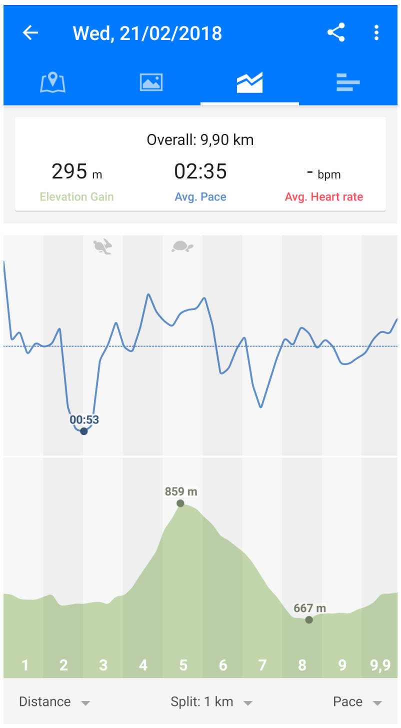


Figure 2.9: The chart which is displayed after the live tracking on the Android application. [Screenshot taken by the author of this thesis from Runtastic.]

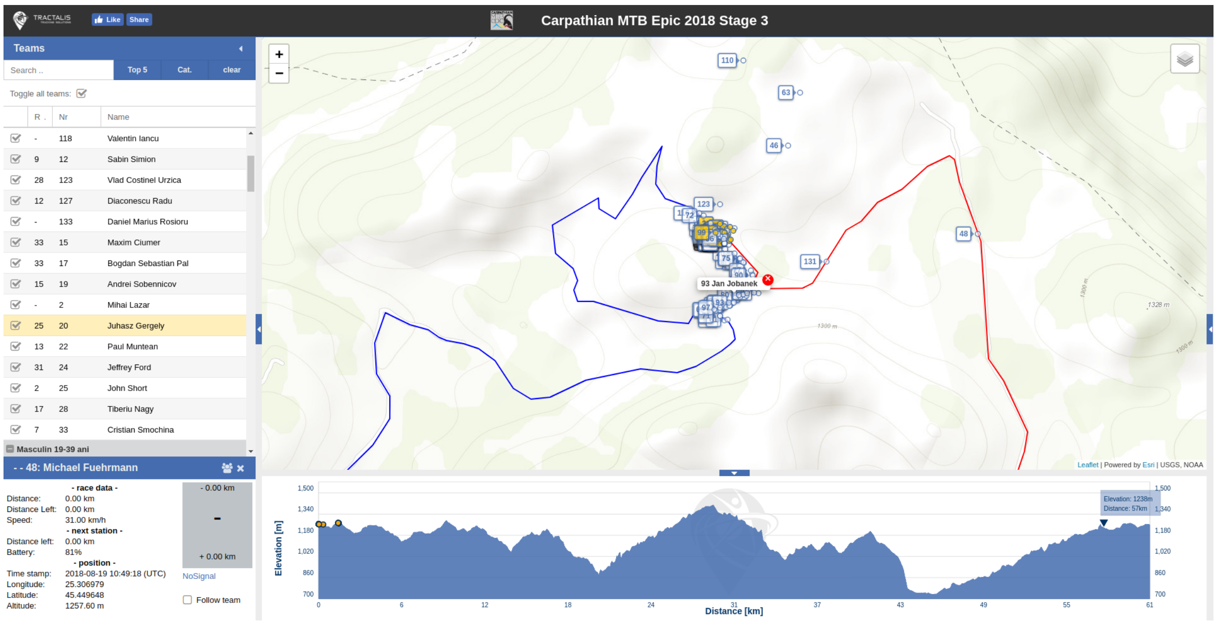
Figure 2.10 shows the live tracking page. On the left side, all athletes of the event are shown. By clicking on an athlete symbol, the display centers on the current location. It also updates the athlete information window in the bottom left corner. This window shows useful statistics about the selected athlete, such as covered distance, distance left to the finish line and the current speed. By clicking on the "Category" button placed above the athlete list, one can open a window (see Figure 2.11) which can be used to display certain categories of athletes, for example, female or male only. Athletes are pictured as the blue circles with the start number right next to them. The application has the feature to display a racetrack, which can consist of multiple segments, which are red and blue lines on the map. However, it is not possible to show the track of an athlete. The altitude profile of the racetrack is positioned below the map. Besides the web application, a tracking app for Android, iOS, and Windows Phone is available.

As mentioned, Tractalis is a good solution for bigger events in which all technical tasks are taken care of by Tractalis. The downside, however, is that the the cost for renting the solution for events could be very expensive (the prices for this solution could not be found). Lastly, this solution is not available for sale but only rent, therefore these charges would be required of the user for each new event.

2.1.3 OpenTracking

OpenTracking[5] is live a tracking, mapping, and timing solution, for sport events. It is a solution much like Tractalis which can be rented. For athlete tracking, it uses the GPS device Queclink GL300[6]. The tracker can send position updates in the interval of one, two, three, five, or ten minutes. The live tracking solution can track up to 1500 competitors per event simultaneously. Acquiring the trackers also gives access to an athlete and event management page. Athletes are assigned GPS trackers which store their start number, name, and nationality, using the online platform. Parts of the live tracking page for an event

[5]Open Tracking, *Event GPS Tracking*. `https://www.opentracking.co.uk/` (visited on 20/03/2019).
[6]Queclink Wireless Solutions, *Queclink GL300*. `http://www.queclink.com/GL300` (visited on 20/03/2019).



Toggle all teams: ✓

	R .	Nr	Name
✓	-	118	Valentin Iancu
✓	9	12	Sabin Simion
✓	28	123	Vlad Costinel Urzica
✓	12	127	Diaconescu Radu
✓	-	133	Daniel Marius Rosioru
✓	33	15	Maxim Ciumer
✓	33	17	Bogdan Sebastian Pal
✓	15	19	Andrei Sobennicov
✓	-	2	Mihai Lazar
✓	25	20	Juhasz Gergely
✓	13	22	Paul Muntean
✓	31	24	Jeffrey Ford
✓	2	25	John Short
✓	17	28	Tiberiu Nagy
✓	7	33	Cristian Smochina

Masculin 19-39 ani

- - 48: Michael Fuehrmann

- race data -

Distance:	0.00 km
Distance Left:	0.00 km
Speed:	31.00 km/h

- next station -

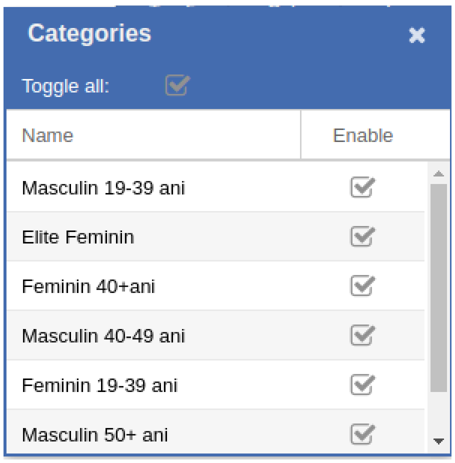
Distance left:	0.00 km
Battery:	81%

- position -

Time stamp:	2018-08-19 10:49:18 (UTC)
Longitude:	25.306979
Latitude:	45.449648
Altitude:	1257.60 m

- 0.00 km
-
+ 0.00 km
NoSignal
☐ Follow team

Figure 2.10: The Tractalis web site to view the live tracking. [Screenshot taken by the author of this thesis from Tractalis.]



Categories

Toggle all: ✓

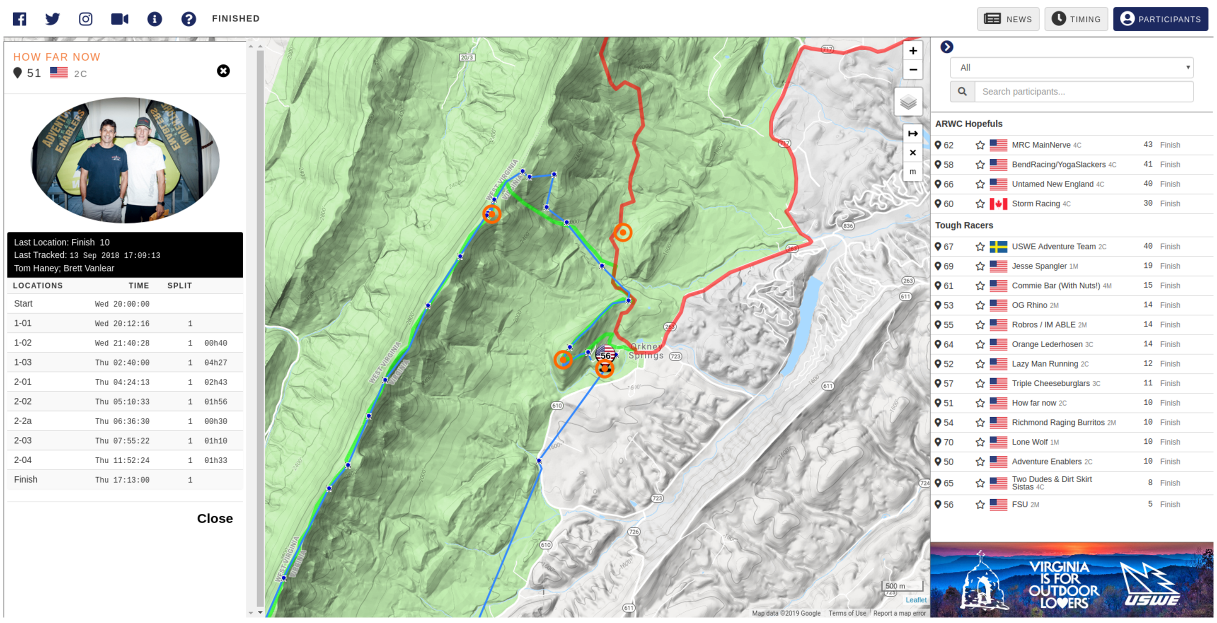
Name	Enable
Masculin 19-39 ani	✓
Elite Feminin	✓
Feminin 40+ani	✓
Masculin 40-49 ani	✓
Feminin 19-39 ani	✓
Masculin 50+ ani	✓

Figure 2.11: A form where the user can filter athletes by certain categories on the Tractalis web site. [Screenshot taken by the author of this thesis from Tractalis.]

can be customized, such as the event info, logos, and social media integration.

The live tracking page consists of three parts: the athlete information window, the live tracking map, and a list that displays all athletes, as shown in Figure 2.12 (from left to right). The event racetrack consists of the green and red lines, which are the different legs of the track. In the shown event the green lines are running legs and the red lines are cycling legs. Coordinates of the racetrack are stored in a GPS exchange format (GPX)[7] file . The orange circles on the map are points of interest along the event track that you can click on to display an information window. The blue line on the map is the track of the selected athlete to distinguish it from the rest, and the current position is a marker with the national flag and start number. It is only possible to show the track of one athlete at once. An athlete can be selected by clicking on the name in the athlete list, placed on the right side. It is also possible to filter the list using

[7]TopoGrafix, *GPX: the GPS Exchange Format*. `https://www.topografix.com/gpx.asp` (visited on 20/03/2019).



HOW FAR NOW
51 2C

Last Location: Finish 10
Last Tracked: 13 Sep 2018 17:09:13
Tom Haney; Brett Vanlear

LOCATIONS	TIME	SPLIT	
Start	Wed 20:00:00		
1-01	Wed 20:12:16	1	
1-02	Wed 21:40:28	1	00h40
1-03	Thu 02:40:00	1	04h27
2-01	Thu 04:24:13	1	02h43
2-02	Thu 05:10:33	1	01h56
2-2a	Thu 06:36:30	1	00h30
2-03	Thu 07:55:22	1	01h10
2-04	Thu 11:52:24	1	01h33
Finish	Thu 17:13:00	1	

Close

ARWC Hopefuls

62	MRC MainNerve 4C	43	Finish
58	BendRacing/YogaSlackers 4C	41	Finish
66	Untamed New England 4C	40	Finish
60	Storm Racing 4C	30	Finish

Tough Racers

67	USWE Adventure Team 2C	40	Finish
69	Jesse Spangler 1M	19	Finish
61	Commie Bar (With Nuts!) 4M	15	Finish
53	OG Rhino 2M	14	Finish
55	Robros / IM ABLE 2M	14	Finish
64	Orange Lederhosen 3C	14	Finish
52	Lazy Man Running 2C	12	Finish
57	Triple Cheeseburglars 3C	11	Finish
51	How far now 2C	10	Finish
54	Richmond Raging Burritos 2M	10	Finish
70	Lone Wolf 1M	10	Finish
50	Adventure Enablers 2C	10	Finish
65	Two Dudes & Dirt Skirt Sistas 4C	8	Finish
56	FSU 2M	5	Finish

Figure 2.12: The OpenTracking web site to view the live tracking. [Screenshot taken by the author of this thesis from OpenTracking.]

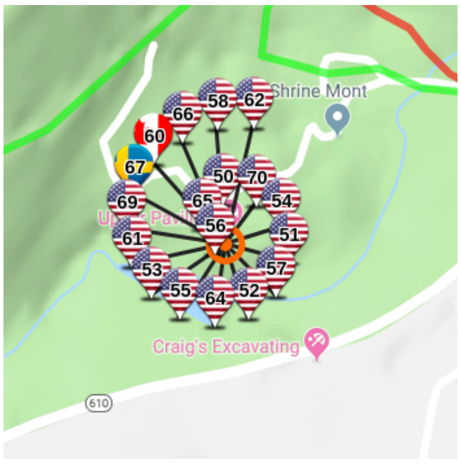
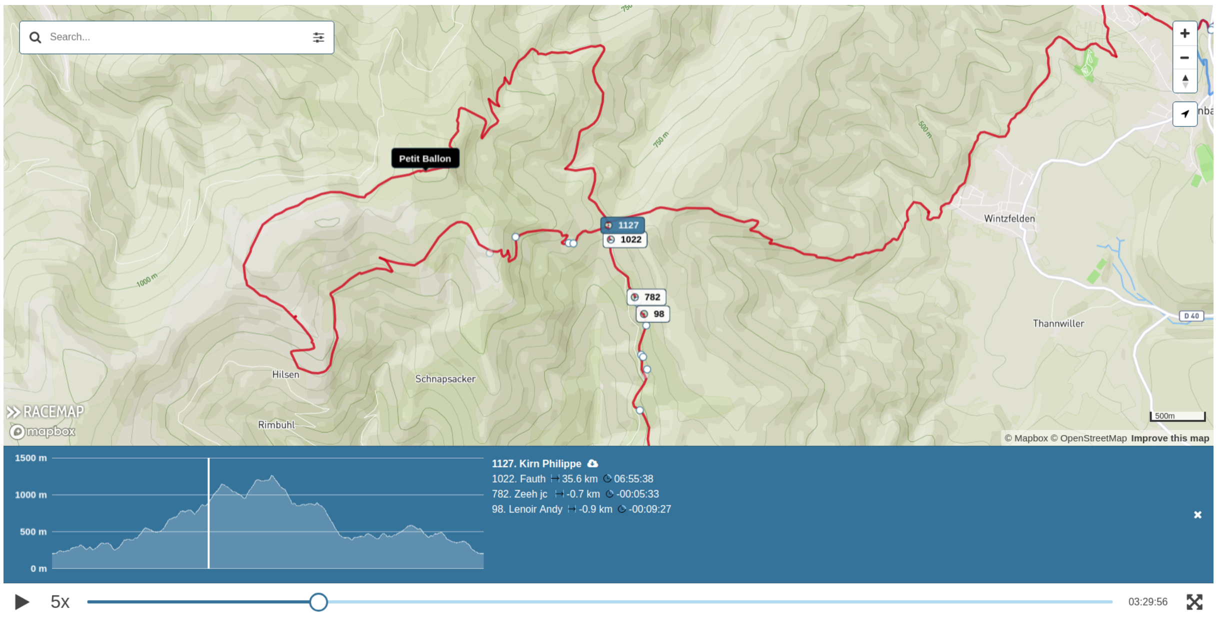


Figure 2.13: Athletes which are close together on the map, clustered into one athlete marker. By clicking on the maker it will show all athletes that are part of the cluster. [Screenshot taken by the author of this thesis from OpenTracking.]

the search bar, or the drop-down list to filter for certain athlete groups. By clicking on an athlete it will also update the athlete info window on the right side. The window displays information such as name, nationality, start number, a picture of the athlete and a timestamp of the arrival at points of interest. A unique feature of the tracking map is the clustering of athletes which are close to each other. The map will only show one athlete marker and by clicking on it, will display the other athletes, seen in Figure 2.13.

The OpenTrack live tracking system is a complete solution to track athletes for sport events. In comparison to other solutions, it is cheaper and can track a large number of athletes at once. The live tracking page offers unique features, such as storing time stamps for every athlete when they pass a point of interest and page customization options. However, the web site lacks statistical evaluations, for example, current speed, pace, or covered distance of athletes.



1127. Kirn Philippe
1022. Fauth 35.6 km 06:55:38
782. Zeeh jc -0.7 km -00:05:33
98. Lenoir Andy -0.9 km -00:09:27

Figure 2.14: The Racemap web site to view the live tracking. [Screenshot taken by the author of this thesis from Racemap.]

2.1.4 Racemap

Racemap[8] is a company based in Germany, specialized in the visualization of live events. Unlike other companies, they offer the possibility to rent the tracking software instead of GPS devices. For payment, a time limited account is created to access the event management page. The web site provides the functionality to use private GPS devices to track athletes. A list of compatible tracking devices is available on their web site[9]. Every added athlete assigned to a GPS tracker can store the name and start number. The number of athletes that can be tracked is limited to the purchased package. However, you can add athletes after the fact by upgrading to a bigger package. To display the racetrack, a GPX file is required.

Figure 2.14 shows the Racemap live tracking page. In contrast to other live tracking web sites, this solution provides less amenities and is less user friendly. The racetrack is the red line on the map. It is possible to place a point of interest along the race track, shown by the black speech bubbles. The white circles with the blue border are the athletes. It is not possible to display the track of an athlete. Clicking on an athlete displays the starting number and opens a blue window on the bottom of the site. This window includes the altitude profile of the racetrack and the current selected athletes. It possible to select multiple athletes at once. The map will always center or rather follow the last selected athlete, identified by the blue colored label. For every other selected athlete (white label), the approximate distance and seconds behind the last selected athlete (blue label) is displayed. However, the live tracking application does not allow you to deselect athletes, or move the map while athletes are selected. The web application must be reloaded to deselect all athletes, which is a not a user friendly feature. It is possible, however, to replay the event in real time or with increased playback speed after or during an event using the replay function.

Concluding, Racemap is a good solution if supported GPS devices already own it. Depending on the number of used GPS devices per event, Racemap can be a more cost effective solution than those of other companies. While the features of the live tracking system are limited, the core functionality is good.

[8]Racemap, *Live GPS-Tracking for Events*. `https://racemap.com/en` (visited on 20/03/2019).
[9]Racemap, *Compatible GPS-Tracker*. `https://racemap.com/en` (visited on 20/03/2019).

2.2 Other Related Work

Besides the live tracking applications described above which mainly focus on tracking athletes during sport events, live tracking and the statistical evaluation of human exercise is also used in other domains.

Smyth and Cunningham [2017] and Berndsen et al. [2017] use the athlete's pace to help them achieve a new personal best for marathons. The goal is to create a race plan for an athlete based on his previous result. The characteristics of the racetrack are also taken into account. In order to do so, the approach uses a large data set consisting of over 600000 records of various athletes from one single racecourse. Each record stores the average pace and the finishing time. Based on the athlete's previous result, the system uses the data set and recommends a tailored race plan. The race plan consists of multiple segments and determines how fast or slow an athlete has to run to reach the targeted time. Other approaches, such as Riegel [1981], Bartolucci and Murphy [2015], and J.Vickers and Vertosick [2016] use other characteristics of athletes to recommend a target time, for example, age, gender, or racing experience.

Providing access to real-time data of sport events is also useful for fans. Since it is not possible to watch every sport live, real-time data allows fans to track events without watching them directly. Wang and Kuzmanovic [2018] discuss how to use real-time data of sport events, for example, football, soccer, or basketball, and present said data to the fans.

Depending on the sport, it is of high importance to select the right tracking device. Edgecomb and Norton [2006] compare technologies which can be used to track players during Australian Football matches. Since Australian Football is a collision sport and the playing field is rather big, the number of viable tracking systems is limited. The tested systems are GPS trackers and computer-based tracking devices which measure the movement distance of the athlete. The study revealed that the recorded distances of both systems always vary from the actually covered distance.

Real-time tracking systems can also be used in other domains. X. Wei et al. [2016] track athletes to predict the next shot location in a tennis match. The goal is to create a system which can emulate world-class tennis players. Tennis players can predict the next move of the opponent based on the speed, location, and angle of the shot. Since professional tennis matches use technologies such as Hawk-Eye or Slamtracker, it is possible to obtain this data. The system uses the recorded data and the thus created probability model of an athlete to predict the type and location of the shot.

A cost-efficient approach to track athletes during live sports events is presented by Pers et al. [2001]. The hardware used to track athletes is a camera mounted on the ceiling, in order to provide a bird's-eye view. In contrast to other systems, only the recorded images from the camera are used to track the athletes' location. The advantage of this method in comparison to other solutions is that the system is inexpensive and barely requires any hardware.

Tracking is not only used for sport events but also for monitoring and improving humans' health. Live tracking devices can be used to monitor the vital signs of soldiers during combat [Patii and Iyer 2017]. Based on the vital sings such as pulse rate, body temperature, oxygen level in the environment and the GPS location of a soldier, the system can act as a lifeguard during combat. The data can be used by other soldiers or paramedics to analyze the well-being of their comrades.
Devices connected to the internet also make it possible to track the health and lifestyle of individuals as presented by Daniel et al. [2011] and Raj et al. [2017]. These tracking devices can monitor vital signs and the activities of humans during the day. The goal is to increase the personal health based on the recorded data. This can be done by sharing the data with doctors or with friends which can empower people to care more about their well-being and personal health.

Chapter 3

Software Development Process

The software development process is a key component for creating the software of the live tracking system. At the start of the project the range of application was not clear, as well as the included features. Therefore it was necessary to gather the requirements. Based on the requirements, it was possible to make major design decisions such as target system, design patterns and software architecture.

3.1 Requirement Engineering

Requirement engineering is important for both developers and stakeholders to gather information about the functionality of the program. Requirements are features the application must include. In these sections, clients or stakeholders refer to the supervisors of this project. The author of this thesis is referred to as the developer.

The requirement engineering process consists of the following tasks extracted from IEEE Computer Society [2014] and Stephen [2015]:

1. Requirement election and gathering
2. Requirement analysis
3. Requirement specification
4. Requirement validation
5. Changing requirements

Those tasks are used to engineer the requirements for the live tracking application.

3.1.1 Requirement Election and Gathering

For the gathering of the requirements, a meeting between the developer and the stakeholder was scheduled. In the dialogue with the clients, the task is to identify the needs and problems they want to solve. It is important to get a clear understanding of the requirements. Due to the limited knowledge of clients on certain topics, they often aren't able to define the problems precisely. During the interview, it is also necessary to filter out requirements that are not important or are not possible to implement. A crucial question during the meeting is to ask where the system should run; whether it should run on a desktop system, a smartphone or a specific operating system. Based on the scope of the software, it can limit the possible solutions. Everything discussed during the interview should be recorded.

3.1.2 Requirement Analysis

After the meeting, the next step is to analyze the records of the interview and extract the requirements. It is also necessary to research the existing system, as stated in Chapter 2. Based on these existing systems it is easier to deduce which features may or may not be possible to implement. Another task is to consider how to implement requirements in the software system or what the software design could look like. All these considerations make it easier to estimate how difficult or time-consuming requirements will be. It is also helpful to ask workmates who may have a better insight in a certain domain in order to get feedback about possible solutions and problems for requirements. Furthermore, analyzing requirements may also filter out redundant features. This happens for example, when the client wants certain features based on an existing system they are using, but because of a new product design, the requirement becomes superfluous. It is also important to analyze the dependency between them, as some features may have a hierarchical structure. Therefore, some requirements cannot exist without implementing the core component, for example, the representational state transfer (REST) interface (see Chapter 5). This also helps to identify key features that are more important for the software system. A common technique that can be used to find solutions or design ideas for requirements is brainstorming. After the meetings, a list was created containing the following points:

Main Page Requirements:

- It should be possible to filter events.
- Active events should be highlighted.
- Upcoming events should be displayed with a countdown.
- Finished events should also be shown on the website.

Athlete Tracking Page Requirements:

- It should be possible to show the racetrack.
- Display information about the event.
- Adding points of interest along the map including a notification service.
- Filtering of athletes that should be displayed on the map.
- Show statistics of the athletes.

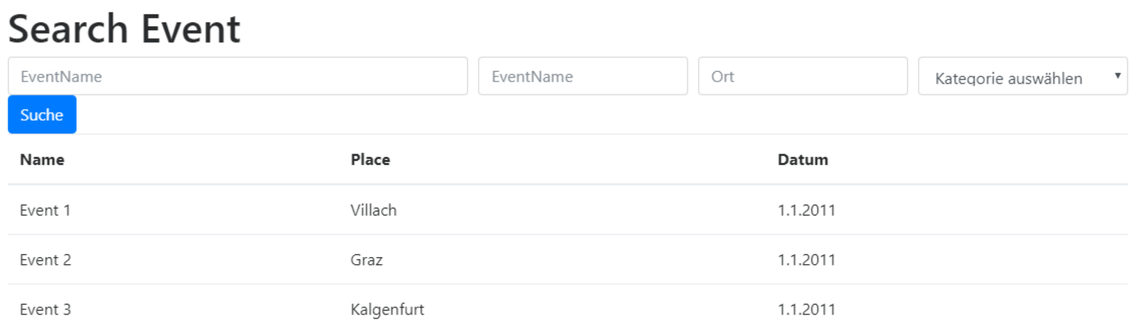
3.1.3 Requirement Specification

This step produces a list of requirements that will be part of the application. It is substantial for the developer and clients to write those requirements down. To represent those requirements, there are many possibilities.

The used approach was *user stories*, which explains how the user interacts with the application and what response the user gets. It also includes prototypes, which are mock-ups of the user interface. The aim is to give the customer a feel for the order of the layout, and the look of certain elements. For the prototyping, Pingendo[1] and Adobe Photoshop[2] was used. After creating the prototypes with these tools, the uses cases for the "Starting Page" and "Athlete Tracking Page" were created.

[1]Pingendo, *Bootstrap 4 builder*. `https://pingendo.com/` (visited on 28/03/2019).

[2]Adobe, *Photoshop*. `https://www.adobe.com/at/products/photoshop.html` (visited on 28/03/2019).



Name	Place	Datum
Event 1	Villach	1.1.2011
Event 2	Graz	1.1.2011
Event 3	Kalgenfurt	1.1.2011

Figure 3.1: Prototype of the event table, which can be used to search and filter for events.

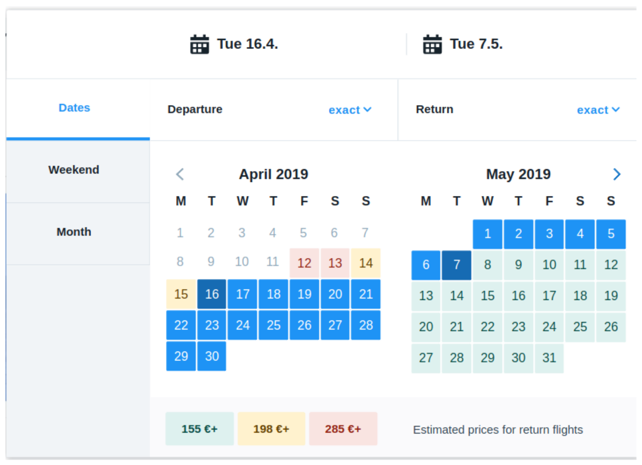


Figure 3.2: Prototype of the date picker to select a date using the calendar window. [Screenshot taken by the author of this thesis from www.kayak.de.]

Use Cases for Starting Page

Event Searching

The user can open the main page and filter the event table using search criteria, such as event name, starting time and location. After entering these criteria, the user clicks on the search button, which updates the table. Figure 3.1 shows a prototype of the table and Figure 3.2 shows a possible date selection bar, which can be used for date filtering. By clicking on an event, the tracking page will open.

Highlighting of Current Events

The starting page displays the current (live) event and the upcoming events. Events are displayed as pictures which include the names of both. Upcoming events are displayed in a slideshow with a countdown timer. Clicking on the pictures opens a the athlete tracking page. Figure 3.3 shows a prototype for the event.

Use Cases for Athlete Tracking Page

Display Track Information

The user can toggle the information window visibility by clicking on the "Information" button in the navigation bar. Depending on the visibility, the button icon changes. Figure 3.4 shows a prototype of the event track information window.

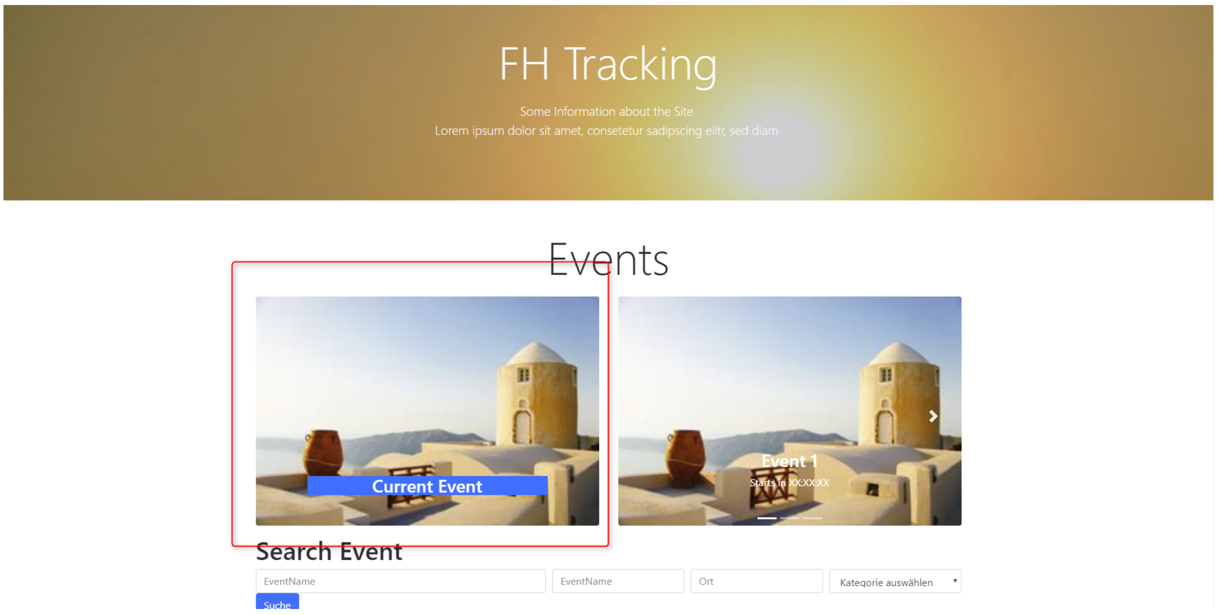


Figure 3.3: Prototype of the carousels, which highlights current and recent events in a image slideshow.

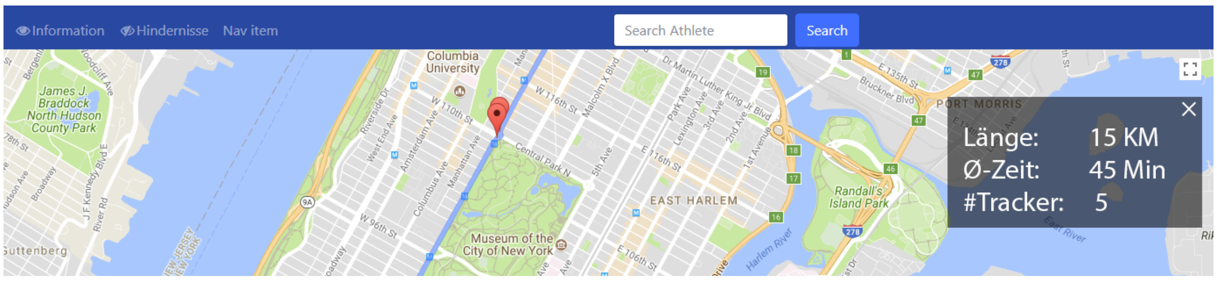


Figure 3.4: Prototype for displaying the information of a racetrack.

Display Point of Interest

The user can add points of interest including an information text. Figure 3.5 shows the mock-up of a point of interest marker on the map.

Selecting Athletes

The user can show or hide athletes by using the drop-down lists placed in the navigation bar. Selecting or deselecting athletes in the drop-down list will toggle the visibility on the map. Figure 3.6 shows the prototype for the athlete selector.

Display Athlete Statistics

The user can toggle the statistics table visibility by clicking on the "Statistics" button in the navigation bar. Depending on the visibility, the button icon changes. Figure 3.7 shows a prototype of the track information window.

Display Notification

The page will create a notification window, if an athlete is near a point of interest. Figure 3.8 shows a mock-up of the notification window.

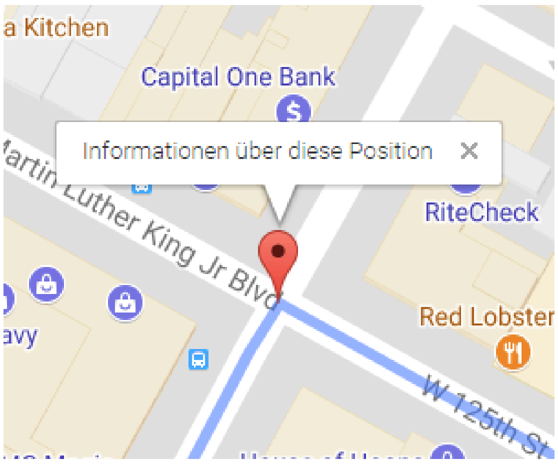


Figure 3.5: Prototype of the point of interest (POI) marker placed on the map. An info window displays information about the POI.

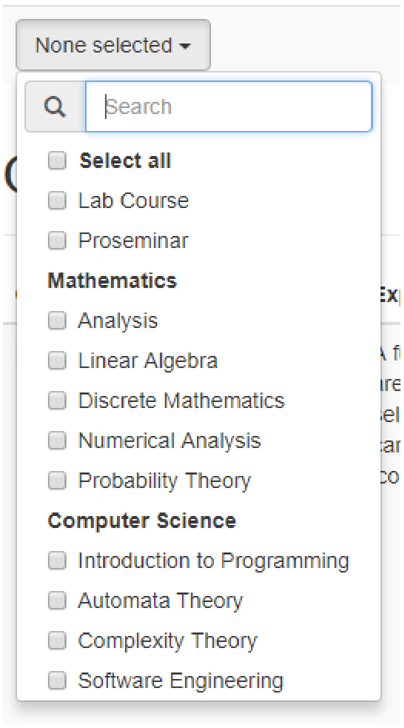
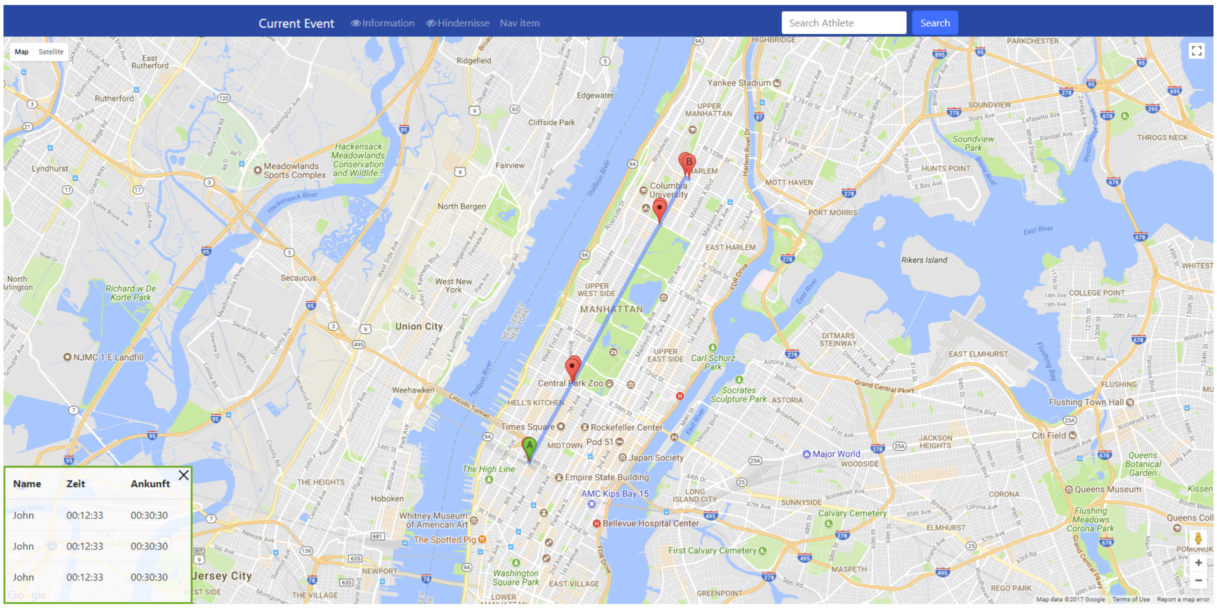
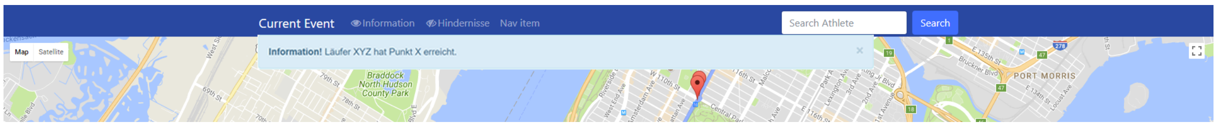


Figure 3.6: A possible selector form which can be used as an athlete selector.



Name	Zeit	Ankunft
John	00:12:33	00:30:30
John	00:12:33	00:30:30
John	00:12:33	00:30:30

Figure 3.7: Prototype of the athlete statistics table which is placed on the map.



Information! Läufer XYZ hat Punkt X erreicht.

Figure 3.8: Mock-up of a notification window to inform the user.

3.1.4 Requirement Validation

The clients will review the created documents, and check if the developers understood the requirements correctly. The client has to check every requirement to see if it describes the task correctly and if any functions are missing.

Requirements that have been removed because of technical difficulties or limitations:

- Filtering of events by using location information such as region or address.
- Creating profiles of athletes and comparing them based on other profiles.
- Creating a leader board for live events
- Storing the fastest time for each event or track.
- Characterization of athletes based on certain information.

3.1.5 Changing Requirements

As in nearly every project, changes are quite common. Requirements turn out to be more difficult to implement than expected, or may be made easier thanks to a certain technique. In most cases, changes are client driven. This may happen after they view a prototype of the application and see something they do not like. Another reason could be that the clients discover a feature they did not think of. Here it is important to agree on what is possible to include into the release and which features must be postponed.

Requirements that were added or changed:

- Add more visibility options for athletes (show marker and/or track).

- Provide a page to upload event tracks.
- Restrict the access to the upload page to user groups.
- Event tracks can have multiple segments.
- Adding a feature to map athletes to roads or the event track.

3.2 Planning the Software Architecture

Software architecture is a set of software structures or a hierarchical order of them, as well as their relationship to one another. The software system can consist of multiple software structures and architecture types. One single software structure does not equal the architecture design of the whole software system.

The structure can have multiple possible characteristics. In a software system, some structures are divided into different implementation units. These groups of structures are called modules and are one of these characterizations. In a bigger software project, common modules are, for example, a database module (see Section 4.2), which handles the connections and data revival, or a user interface module, which includes all forms of data and graphics (see Chapter 6). Modules can contain a large amount of structures and it is possible to decompose them again into another set of modules (see Sections 4.3, 4.4, 4.5, and Chapter 5). The second characterization is a dynamic structure. Until the program is running on the system, the software only consists of the source code. These structures only exist during the run-time of the program, therefore they are called dynamic structures. As an example, a program creates a list of users, which are stored in a database. Neither the amount of users nor their properties are part of the program until it runs on the system and can fetch the data from the database.

Choosing the right software architecture is an important procedure for every project. A developer can choose to create the code "on the fly" without any clear architectural concept. It is arguable that instead of wasting a lot of time analyzing requirements and designing an architectural concept, it would be better to start coding right away. Following this approach, the productivity in terms of produced lines of code would be much higher at the beginning. In the end however, the code is not very easy to maintain. Without choosing a proper software architecture, it becomes harder to implement changes. As the word "soft" in the word software already indicates, modifying software is meant to be easy. Dealing with changes is a common task in software development. In practice, stakeholders often change their mind about a user case, or what may also occur is that there are some changes due to switching to a different library, which is a key part of the code. As a study shown by Martin [2017] indicates, it becomes harder for software developers to be efficient, without using a proper software architecture.

The next step after analyzing the requirements is to design the software. The design process is divided into two parts, the high-level design and the low-level design. The high-level design provides a more abstract point of view about the whole software, for example, the client-server architecture. Based on the requirements, the developer has to analyze the needs of the program. Is it possible to split the application in different parts as to what purpose each part has, such as statistics, mapping algorithms, chart creation or the different controllers of the client.

3.2.1 Security

Security concerns are not as important regarding this project. The main goal is to create a live tracking software for events. In most cases, the client-server communication narrows down to data exchange. The data does not contain any sensitive data, such as personal information or stored passwords. In most cases, the transferred data is publicly accessible via the REST interface, containing geographic coordinates of global positioning system (GPS) trackers or information to the event. To respect the privacy of athletes, only pseudo names and the gender is stored.

Another security concern is the limitation of access. It should be possible for groups like admins or event managers to access certain sub-domains or functionalities. For example, only admin and user should be able to upload a file that contains the track coordinates. To solve this problem, Java EE servers like Jetty[3] provide the possibility to grant certain user roles access to predefined URL patterns.

The requirement to upload files for events produces another potential security risk. A user could upload malicious files to exploit the system, therefore checking the syntax of them is necessary.

3.2.2 Hardware

The client's request was to create an application that is accessible from nearly every device with an internet connection and a display. Therefore, the hardware specification consists of two parts. One part is the client, where the user interacts with the application (see Chapter 6). The second part is the server that is responsible for the data transaction and other computations (see Chapters 4 and 5).

For the presentation of the client's end, a possibility would be to create a native application for selected operating systems. Researching the internet, the most used operation systems in October 2017 was Microsoft Windows with 42.76% in first place, followed by Android phones with 35.22% and in third place Apple phones with iOS with 15.69%. Other systems such as Mac OS, Windows Phone, Chrome OS and Linux have a combined 5.78%[4].

Creating a native application for every operating system may increase the performance of the application. The reason for this is each operating system has its own integrated development environment (IDE), to create applications tailored for the operating system. Although a native application may perform better, not all IDEs uses the same programming language. Therefore, it is not guaranteed that libraries which are used for one platform are compatible or available for others. For this reason it would be a lot of work to find a good solution for every target system and keep consistency. Consider all these obstacles, a native application for six different operating systems would be overkill and not manageable in the limited time available for the project.

Since nearly every device with an internet connection can access the web sites via a web browser, a possible solution was to create a web application, also known as web app. A web app does not depend on a certain operating system or what process architecture a client uses. The only program users needs to display the web application is a "modern" web browser that supports JavaScript. Web applications are based on the client-server model. A client is the web browser that displays the website on the users devices. Today, web apps consists of four major components. The hypertext markup language (HTML) [Smith 2017] file is responsible for the structure of the application, which consists of text, images, hyperlinks, or other HTML elements. To define the representation of a website, cascading style sheets (CSS) [Rivoal et al. 2019] are used. This file defines the look, layout, and font of the content. The goal is to separate the content, which is in the HTML file, and the styling, which is stored in the CSS file. The third component is JavaScript. This technology makes the web application more responsive. JavaScript supports dynamically adding content or manipulating existing content without reloading the complete website. To load the content dynamically, JavaScript is used in conjunction with asynchronous JavaScript and XML (AJAX). AJAX communicates and exchanges data with the web services. In most cases, every web browser can process these files and render the web application independently from the operating system. Another advantage while using a web app is that the client does all the computation to create the web page. The only service the server must provide is to make the files accessible over the internet. The second task of the server is a service to retrieve data from the server. The service which is implemented on the server is discussed in Chapter 5.

[3]Eclipse, *Jetty - Servlet Engine and Http Server*. (Visited on 20/03/2019).
[4]StatCounter, *Operating System Market Share Worldwide*. `http://gs.statcounter.com/os-market-share` (visited on 28/03/2019).

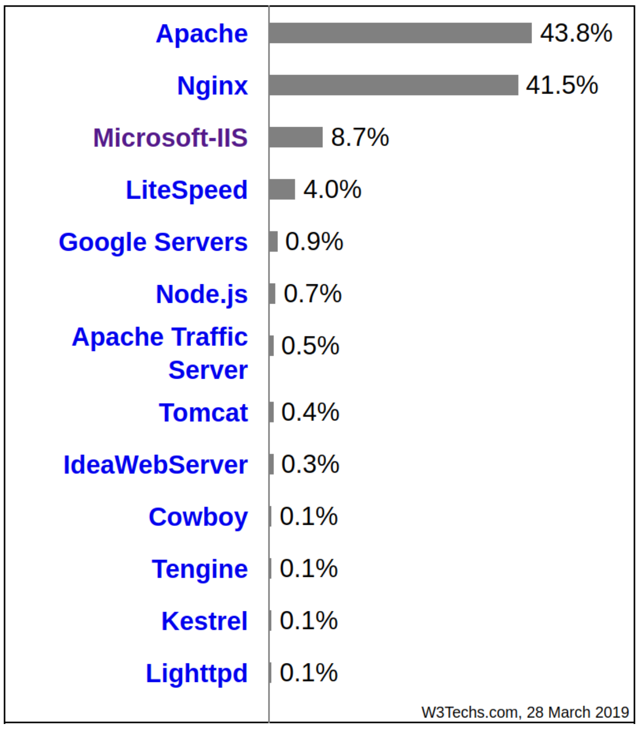


Figure 3.9: Usage Statistics and Market Share of Web Servers from March 2019. [Screenshot taken by the author of this thesis from W3Techs.]

The web browser was chosen as the target system because it does not limit the software to a certain operating system or hardware configuration. The second hardware requirement is to create a server running on a Linux Debian 64 bit system. Linux is a developer friendly platform because it does not limit the possibilities of solutions, as nearly any application is compatible with it. In fact, even when applications may not be, Linux can edit the documents to be compatible, which cannot be said of many other operating systems. Therefore, the amount of possible web servers is large. To get a better overview of which servers are used online,[5] gives a statistical analysis of the usage of web servers for websites, shown in Figure 3.9.

The idea behind searching for popular servers is that the support for those services is higher and documentations for solutions are easier to find. In addition, the update cycle for these servers is more frequent.

3.2.3 Web Server and Framework

After the decision to build a web application, it was necessary to research possible solutions. The first solution is to use a web framework. Popular web frameworks that are widely spread include:

- Spring Framework
- NodeJS
- Ruby on Rails

[5]W3Techs, *Usage Statistics and Market Share of Web Servers, March 2019*. `https://w3techs.com/technologies/overview/web_server/all` (visited on 28/03/2019).

3.2.3.1 Spring Framework

Spring Framework[6] is an application framework written in Java. It uses the Java Platform Enterprise Edition (Java EE)[7], as many other web servers do. The Spring framework helps to build java applications and deploy them as a web application based on the Java EE. The spring framework provides many features that help to create a web application.

- The spring framework has an embedded run-time environment, which uses a Java EE web server, such as Tomcat or Jetty.
- It uses aspect-oriented programming (AOP). AOP means that features such as security, logging, or user management, are separated from the business logic.
- User management and login service.
- Own implementation Java Database Connectivity (JDBC) application programming interface (API). This module converts tables from the database to Java objects using SQL queries.
- Spring Framework has a built in model-view-controller (MVC) framework. The controller is written in Java and handles the mapping. The view and the models are defined with JavaServer Pages (JSP).
- Has an integrated testing suite.
- Rest Point creation is possible.

3.2.3.2 NodeJS

NodeJS[8] helps to create web applications and is written in JavaScript. In contrast to the Spring Framework, NodeJS runs on Google Chrome's JavaScript Engine (V8 Engine). For that reason, NodeJS is available for nearly every platform. The application created for NodeJS must be written in JavaScript to execute them with NodeJS. NodeJS has following properties:

- NodeJS is a very lightweight framework with a size of under 3 mega bytes.
- All APIs that call on the NodeJS server are asynchronous and event driven. This means that if the program executes an API call and has to wait for the data it does not block other calls. If the data is available, the server fires an event to return the data and finish the API call. This method is called non-blocking.
- NodeJS only runs on a single thread using event looping. The non-blocking method explained above makes the server highly scalable compared to other servers, which can only run on a limited numbers of threads.
- Applications created for NodeJS do not buffer any data. The data output returns in chunks.
- Possible to create RESTFul APIs.
- Web browsers do not allow access to the file system using JavaScript. However, NodeJS provides this feature.
- Implementing MVC only with plugins possible.

[6]Spring. `https://spring.io/` (visited on 28/03/2019).

[7]Oracle, *Java Platform, Enterprise Edition (Java EE)*. `https://www.oracle.com/technetwork/java/javaee/overview/index.html` (visited on 20/03/2019).

[8]Node.js. `https://spring.io/` (visited on 28/03/2019).

3.2.3.3 Ruby on Rails

Ruby on Rails[9] also known as Rails, is an application framework written in Ruby. Rails runs server-side and provides an MVC framework. It encourages the developer to use state-of-the-art web standards. For data transmission JavaScript object notation (JSON) or extensible markup language (XML) is used, and for the user interface, HTML, CSS, and JavaScript.

Ruby on Rails also has default structures to map database records form a MySQL or PostgreSQL to a view container. Ruby also includes the following features:

- Ruby on Rails has already been in use for 13 years. Because of that, many documentations and books for research are available.
- Ruby has a rich support on libraries. As a result that Ruby on Rails is in development since 2005, there are thousands of plugins that can be used for the creation of web applications.
- Rails includes an AJAX library with many functionalities out of the box. It automatically produces an AJAX code from a Ruby code. The JavaScript files use the generated AJAX code to receive data from the server and load it into the view.
- Rails uses the convention over configuration (CoC) principle software design paradigm. This means using programming conventions to configure the application instead of configuration files.
- An advantage of the CoC is that Ruby on Rails automatically maps tables to classes and rows to objects. Therefore, objects do not need additional linking in a configuration file.

3.2.3.4 Conclusion for Web Framework

A big advantage to using existing frameworks is that documentations and lectures are already available, which makes the creation of a new application easier. All three frameworks provide the possibility to use the implemented MVC solution for creating web applications. Despite the fact that these frameworks use different platforms or interpreters to run, they all support the implementation of a REST interface. Another positive characterization is that all of these frameworks offer to use plugins, which help solve problems or make some tasks easier to implement. Using a framework, however, can also have some disadvantages. Creating an application with a new framework always takes time to get used to it. Furthermore, external developers, who want to enhance the application may have to double their effort. They would have to get to know how the framework and the generated code works before starting. Another downside is that web frameworks can be inconsistent and still have bugs, which makes it harder for developers to get used to the system [Ocariza et al. 2015]. Another disadvantage is that a web framework might be overkill. Using a framework might offer a lot of functionalities and features, but also unnecessary ones. Not all features are useful for a certain application and they can make it more complex than it should be. It can also be the case, that frameworks does not includes features which are necessary for the development [José Ignacio et al. 2008]. Although all frameworks offer the possibility to implement a REST interface, they are not always used in the proper way. As well as Ruby on Rails, Spring, and NodeJS use a controller which builds the view on the server's side, and not via JavaScript on the client's side. The controller also does not use the created REST interface to receive the data. Using two interfaces that produce data is redundant and not a good solution. If the model or the database schema changes, both interfaces must change their implementation. Furthermore, web frameworks use the REST interfaces most of the time to return a HTML page instead of the data that is needed, which is not a good approach [Vosloo and Kourie 2008]. An important part of this application is to visualize the athlete on a map. For that reason, an interactive

[9]Hansson, *Ruby on Rails*. `https://rubyonrails.org/` (visited on 28/03/2019).

map library is needed, such as Goolge Maps[10] or Open Street Map[11]. The components on the map, for example, the event track or the athlete path cannot be dynamically created using the MVC concept provided by these frameworks. Frameworks also dictate how the system runs. It is difficult to include features of another framework into an existing one. By not relying on frameworks it is possible to create a application and add the features that are really needed from libraries [Kazman et al. 1998]. Although web frameworks are constantly updated, they still have security gaps, for example, cross-site scripting [Peguero et al. 2018]. Due to these reasons, a framework for this application would be unreasonable and it would be better to create a more lightweight framework from scratch.

3.2.4 Client-Server Model

After the decision to create a web application, it was clear that a client-server architecture had to be used. The client-server architecture consists of two parts. The server, which offers some services, and the client, which uses the them. Decoupling these two components makes it easier for the developer to separate concerns of the application. In many cases, the client is responsible for the graphical representation of the program. The server offers the functionality to access data from a database, adding new data, or deleting them. Implementing the database directly on the client side is not a good approach, as this would lead to every client having their own database with a different data set. An approach where the server is directly implemented on the client's side is shown in Figure 3.10.

By adding a layer between the client and server, it helps to separate those two components. Therefore, multiple clients can use the same data set. The layer that separates client and server is called a tier. In the two-tier architecture, one tier separates the client from the server, shown in Figure 3.11.

The server transfers the data to the client via a network such as the internet, a local area network or intranet. Using the two-tier system is a better solution than using the first one, but still not ideal, therefore this project uses a three-tier architecture.

Although the two-tier architecture decouples the server from the client, they are still very close. It is also possible to support multiple clients, but changing the data representation on the server, the client has to make changes too. Another problem may be that not all clients can process a certain data representation. Therefore, adding an additional layer is necessary. The layer that connects the database with the client via a network is called middle tier, shown in Figure 3.12.

The task of the middle tiers is to offer an interface for the client to exchange data with the server. This application uses a REST interface, as discussed in Chapter 5. A further task is to map the database to the REST interface. The client-server architecture was also useful to make the development of the project easier. Each part of this architecture can be developed separately without blocking the development of one another. At the start of the project, the database does not have any data, but on the middle tier it is possible to start creating a logic to connect to the database server. A further plus point is that you can create dummy data sets using fake data, making it easier to test certain scenarios. Adding those dummy data sets does not have any effect on the data stored in the database. The web application can still be accessible while changing the implementation of the layers. On the client's side it is possible to create the user interface without having the database finished or running all the time. The middle tier consists of multiple tiers which is responsible for several tasks:

- Data Access (see Section 4.2)
- Creating statistics (see Section 4.3)
- Mapping the athlete track (see Section 4.4)

[10]Google, *Google Maps*. `https://maps.google.com/` (visited on 20/03/2019).
[11]OpenStreetMap. `https://www.openstreetmap.org` (visited on 20/03/2019).

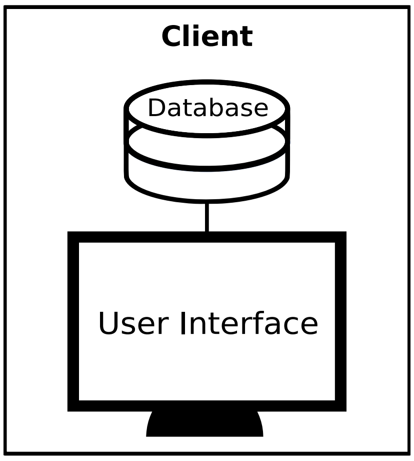


Figure 3.10: The data and services are part of the client and no connection to a server is needed.

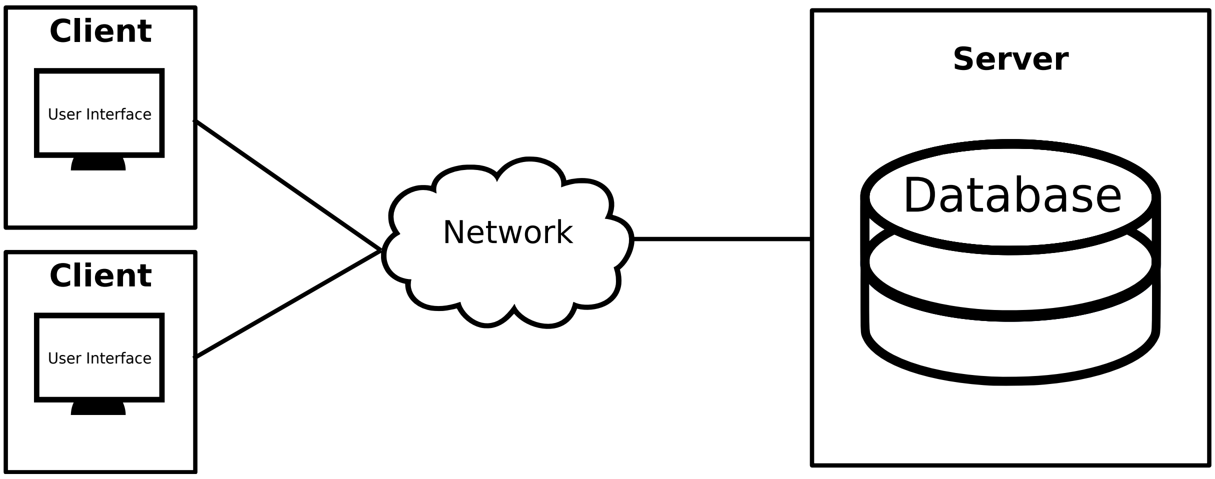


Figure 3.11: The client-server two-tier architecture separates the client from the server.

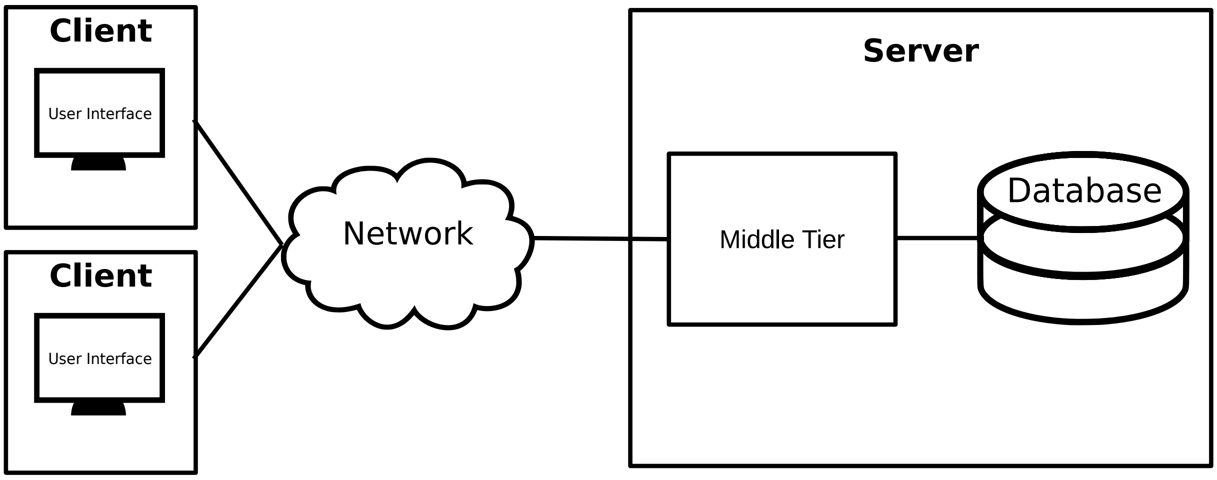


Figure 3.12: The client-server three-tier architecture separates the client from the server with a middle tier. [Image drawn by the author of this thesis.]

- Creating data and structure for charts (see Section 4.5)
- REST interface (see Section 5.7)

Chapter 4

System Architecture and Overview

After the requirement engineering process and specifying the higher level software design, described in Sections 3.1 and 3.2, this chapter describes the used technologies and gives a more detailed insight of the low-level software design of the server. It also discuses the design decisions of important implementations and the pros and cons of the used approach.

As mentioned in Section 3.2.4 the live tracking application uses the client-server pattern. Figure 4.1 shows a more detailed view of the software architecture. The server is written in Java and consists of multiple layers which have different task. The data access (see Section 4.2.2) layer is responsible for connecting to the database and providing an interface for other layers, such as athlete statistics, athlete mapping, chart creation and the representational state transfer (REST) interface. The athlete statistics (see Section 4.3) layer calculates statistical values to track the performance of athletes during a live event. Athletes' performances can also be visualized using charts, as described in Section 4.5. The chart layer creates a software structure as output, which is used by the client to render a chart for the user. The task of the athlete mapping layer is to improve the mapping of athletes on the map. The layer includes multiple algorithms which can align the athlete track to roads or on the racetrack. The produced data of these layers are accessible for the client and other users via the REST interface (see Section 5.7). The client uses the data of the REST interface to create the view for the start page, the athlete tracking page and the upload page, as described in Chapter 6. The client is a web application and consists of hypertext markup language (HTML), cascading style sheets (CSS), and JavaScript files provided by the server, and can be displayed in any browser.

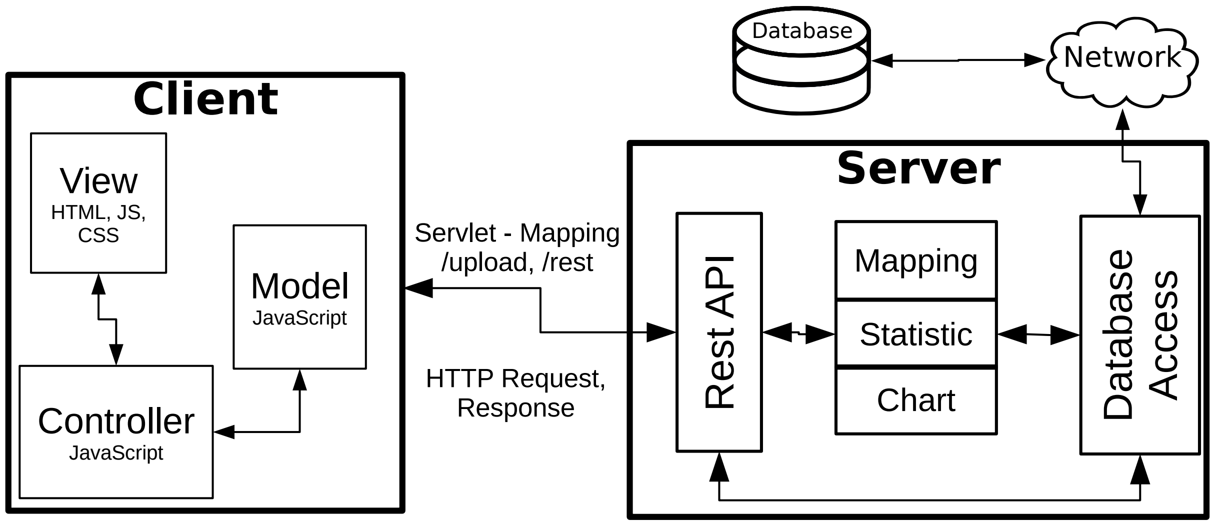


Figure 4.1: More detailed diagram of the software architecture.

4.1 Used Technologies

4.1.1 Jetty

Eclipse Jetty[1] is an Hypertext Transfer Protocol (HTTP) server written in Java. The Jetty Java server is a free, open source project developed by the Eclipse Foundation. Like any other Java (Web) servers, Jetty supports JavaServer Pages (JSP) and the Java Servlet API. Supported protocols are HTTP and WebSocket, which are important for sending data over the internet. The server provides access to the resources of the web application and the REST interface.

4.1.2 MySQL

This project uses MySQL[2] as a database management system and it is a free open source software. It plays a major role for data storage, data evaluation and as a communication interface between server and client.

4.1.3 Java Servlet

A server uses Java Servlets[3] to process client requests, and create dynamic or static resources. In this project, Java Servlets are used to map static or dynamic resources such as files of the web application or REST interface to a specific URL patterns. Servlets can also restrict specific URL patterns to particular user groups, for example, admins and event managers. For instance, the path to the REST interface always starts with `/rest`. The upload page has the static path `/upload` and restricted access.

4.1.4 Java API for RESTful Web Services Rest (JAX-RS)

The Java application programming interface (API) for RESTful Web Services (JAX-RS) supports the creation of REST web services on the server. This library uses Java Annotations [Gosling et al. 2019] to create RESTful interfaces. With these annotations, the output of methods is bound to a specific URL,

[1]Eclipse, *Jetty - Servlet Engine and Http Server*. (Visited on 20/03/2019).
[2]MySql. `https://www.mysql.com` (visited on 20/03/2019).
[3]Oracle, *Java Servlet Technology Overview*. `https://www.oracle.com/technetwork/java/javaee/servlet/` (visited on 20/03/2019).

which a client can access using different HTTP methods. The server uses the Jersey[4] library to create the REST interface, which is compatible with the JAX-RS API.

4.1.5 JavaScript and ECMAScript

JavaScript (JS) is a scripting language mostly used by web applications. JS can manipulate or create new content for a web page without reloading the whole site. Besides HTML and CSS, JS is the most important technology to create modern web sites. In this application, JS is used to transfer data between client and server. For example, during a live event the location data of an athlete updates every ten seconds. Reloading the whole page every time to update the data would not be efficient, as reloading takes many resources both from the client and server. The client must re-render the map and request data from the server again. The server must access the database, compute them and send them to the client. JS is mostly used in the controller of the model-view-controller (MVC) pattern of the client and is a mediator between the model and the view (see Chapter 6).

ECMAScript (ES) was created to standardize JS. Companies that created web browsers tented to use their own standardization for client site scripting earlier on. The European Computer Manufacturers Association (ECMA) goal was to create a standardized specification for the scripting language. The standard has been designed and developed with the two most important companies at that time, Microsoft, and Netscape. The standards ECMA-262[5] and ISO/IEC 16262[6] were created in June 1998. Nowadays, almost all browsers and devices support nearly every feature that is part of the ES standard. The current version of ES is ECMAScript 2018[7] and it supports features such as accessing REST and asynchronous iteration.

4.1.6 JavaScript Object Notation (JSON)

The JavaScript object notation (JSON) [Bray 2017] format is a compact data format consisting of one or multiple key-value pairs called property. Values can use predefined data types, such as Boolean values, numbers, strings, arrays, and objects. The data can be arbitrarily nested as well; an array can hold multiple objects, for instance. JSON is essential for storing and accessing data using JavaScript. Listings 4.1 shows an example of a JSON file.

4.1.7 Gson

Gson[8] is an open source Java library to serialize Java objects. Gson is used by the REST interfaces to generate the JSON output. To transfer objects, they need to be converted into a target structure or object. This is called a data transfer object (DTO). Gson makes the usage of DTOs unnecessary because the mapping of the model (see Section 4.2.2) to the output format (JSON) is done directly. For the mapping, a so called serialized adapter must be implemented to map the object values to target format.

[4]Jersey, *RESTful Web Services in Java.* `http://oracle.com/technetwork/articles/javase/index-140168.html` (visited on 01/03/2016).

[5]Ecma, *ECMAScript: A general purpose, cross-platform programming language.* `https://www.ecma-international.org/publications/files/ECMA-ST-ARCH/ECMA-262,%201st%20edition,%20June%201997.pdf` (visited on 20/03/2019).

[6]ISO, *ISO/IEC 16262:1998.* `https://www.iso.org/standard/29696.html` (visited on 20/03/2019).

[7]Ecma, *ECMAScript 2018 Language Specification - ECMA-262.* `https://www.ecma-international.org/publications/files/ECMA-ST/Ecma-262.pdf` (visited on 20/03/2019).

[8]Google, *A Java serialization/deserialization library to convert Java Objects into JSON and back.* `https://github.com/google/gson` (visited on 20/03/2019).

```
{"event": {
  "id": "1",
  "name": "Ironman",
    athletes": [
      {"name": "Max Mustermann", "age": "25"},
      {"name": "Maria Musterfrau", "age": "22"}
    ]
}}
```

Listing 4.1: An example of a JSON file.

4.1.8 jQuery

jQuery[9] is a very efficient and fast JavaScript library that makes tasks such as event handling of the user interface, asynchronous JavaScript and XML (AJAX) communication or making changes on the document object model (DOM) of a web site much easier. jQuery is necessary because not all browsers have the same API implementation or they implement certain features differently. All functionalities of jQuery work on nearly every browser and mobile device and are constantly tested[10]. This library is also required for using Bootstrap and their plugins. The most essential functionalities of the jQuery library are:

1. Performing an HTTP request with AJAX.
2. Event handling of HTML elements and user interaction.
3. Loading dynamic or static content into the web page.

To retrieve data from the REST Interface, the application has to perform an asynchronous request using JS. Performing this task only using JS would add extra code to perform error handling and other tasks. This is why it is a better solution to use the built-in AJAX functionality of jQuery. The library makes server requests much easier and automatically converts the received data from the server into a JSON format. Using AJAX calls does not block other code while waiting for a response from the server. Section 6.3.1 gives a more detailed explanation.

When the user interacts with the web page, for example, by clicking on a button, dragging an element, or changing the value of an HTML form, the browser triggers an event. Handling those events and dynamically changing elements makes web pages more responsive. jQuery can select unique identifier (ID) or a group of similar HTML elements (CSS classes) and assign them a task for a specific event. Every HTML element can trigger certain events called document object model events, which are standardized by the World Wide Web Consortium [Kacmarcik and Leithead 2018]. An example of the usage of event handling is the drag and drop function of the athletes' statistics table. The user drags a column of the table and places it in another position. The order has changed and the "on drop" events fires. The browser calls the assigned function, which then changes the drawing order of the athletes on the map.

Adding new content to the web page, changing elements or completely removing them, is one of the key features of jQuery. Manipulating the content is often done after an AJAX request or an HTML event. jQuery has built-in functions to manipulate HTML elements, for example, adding, removing, or toggle styles. The library also supports inserting content inside, outside, or around an element in the

[9]jQuery. `https://jquery.com/` (visited on 20/03/2019).
[10]*jQuery Browser Support.* `https://jquery.com/browser-support/` (visited on 20/03/2019).

	Extra small <576px	Small ≥ 576px	Medium ≥ 768px	Large ≥ 992px	Extra large ≥ 1200px
Max container width	None (auto)	540px	720px	960px	1140px
Class prefix	.col-	.col-sm-	.col-md-	.col-lg-	.col-xl-
# of columns	12				
Gutter width	30px (15px on each side of a column)				
Nestable	Yes				
Column ordering	Yes				

Table 4.1: The five breakpoints of Bootstrap and their properties.

DOM. jQuery can also change the behavior of HTML elements. For example, a user manually updates the statistics table using the update button. The button is then disabled for several seconds to prevent unnecessary server requests.

4.1.9 Bootstrap

Bootstrap[11] is a free open source front-end web framework using HTML, CSS, and JS, developed from employees of Twitter. This framework supports building responsive and mobile-first front-ends for web sites. Today various devices are used to view web pages. Devices such as smartphones, tablets, and laptops have different display resolutions, and a static layout is not ideal to create a proper view for all devices. Bootstrap helps to solve this problem. It uses a grid system, which has five breakpoints that set the maximum width of containers, according to the width of the current display resolution. Based on these responsive breakpoints, the content of the web page is displayed differently without creating a separate layout. If the width of the browser changes, the break point changes as well and the framework automatically re-renders the web page according to the new break point. Table 4.1 shows the five breakpoints, the width of the containers and other properties according to each breakpoint.

To use Bootstrap on any web page, the following files must be included in this order:

1. `bootstrap.css`: Includes all styles and behaviors of the layout such as grid system, components, color scheme and so on.
2. `jquery.js`: Includes the jQuery library see Section 4.1.8.
3. `popper.js`: Helps to correctly display and position drop-downs, tool-tips and pop-overs.
4. `bootstrap.js`: Necessary to trigger events, animations, re-render components and other responsive behavior.

With those four files above, a Bootstrap grid system can be created. Inside the `<body>` of the HTML document, a `<div class="container">` container is placed. This container holds one or multiple rows, which are `<div class="row">` elements. Each row can contain up to twelve columns (`<div class="col ">`). Grid-Columns have many functionalities:

- Width: The width of a column can be fixed, automatic, or equally distributed. The column width is a concatenation of the break point prefix and the number of columns. for example, `col-3` defines a cell which uses the width of three columns from the twelve column grid. If no column width is defined, the framework distributes the width equally for all columns in a row. In case a row consists of columns with fixed and non-fixed width, the width of the non-fixed is equally distributed to fill all twelve columns.

[11]Bootstrap, *The most popular HTML, CSS, and JS library in the world.* `https://getbootstrap.com/` (visited on 20/03/2019).

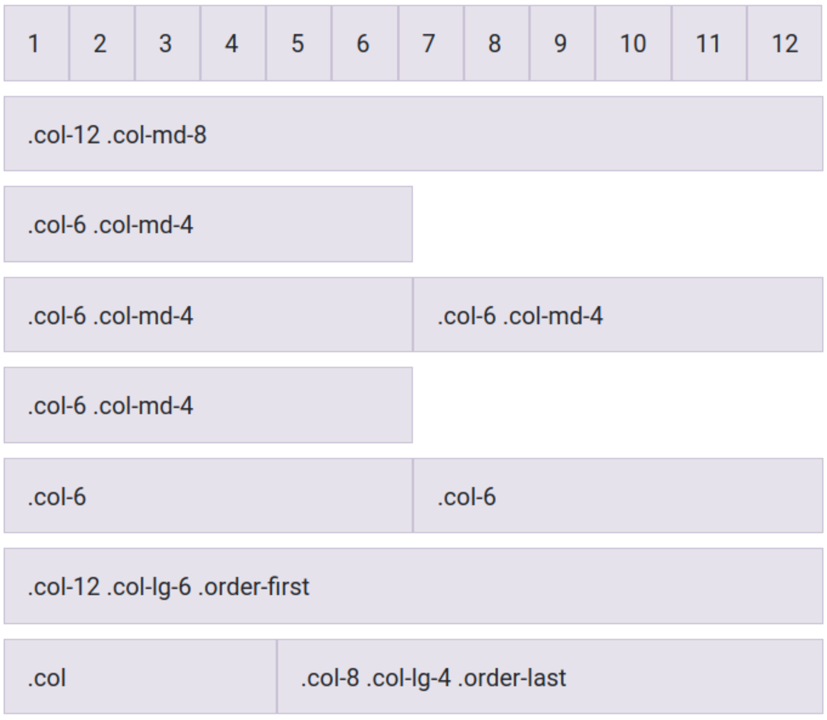


Figure 4.2: The rendered web page for devices with a screen width smaller than 992 pixels.

- Ordering: The order of columns can be changed without restructuring the layout. To reorder columns, the order class `.order-` must be set and can be numerated from one to twelve across all five grid-tires. A column with a lower number (`.order-2)` will be ordered before a column with a higher number (`.order-` 6). The reorder functionality makes it much easier to implement a right to left support, for instance.

Listing 4.2 gives a simple example on how to use breakpoints and the CSS classes of Bootstrap. Inside the `<div>` container there are three rows. The first row consist of two columns. If the width is smaller than the width of the medium break point (768 pixels), the row only holds the first column. The second column is pushed into a new row because there is no more space available and it takes half of the row's width. If the display width is greater than the medium break point, the row has two columns. The first column uses 8/12 of the row width and the second uses 4/12.
In the second row, all columns have 50% of the width on smaller devices, and 33.3% on larger ones. It is also possible to replace `col-md-4` with `col-md` because Bootstrap can automatically assign the same width to all three columns.
In row number three, a column always takes 50% of the width for all breakpoints.
Row number four gives an example of the reordering functionality and auto-width of a column. The first column takes 66% of the width on smaller devices and 33% on devices with a display width more than 992 pixels. Setting the CSS class `.order-last` will always display the first column as last in this row. The second column takes the full width on smaller devices and 50% on lager ones. This column will always be the first column in this row, because `.order-first` is set. The third column uses the auto width function provided by Bootstrap. The framework calculates the maximum available space in a row and assigns it to the column. In the case of smaller devices, the width will be 33% of the row because the first ordered column pushes both in a new row. On lager displays, the column span is two. Figure 4.2 shows the output of the HTML code on devices with a smaller screen and Figure 4.3 shows the output on lager displays.

```html
<html>
<head>
...
</head>

<body>
...
<div class="container">
<!-- Stack the columns on mobile by making one full-width and the other half-width
    -->

<div class="row">
  <div class="col-12 col-md-8">.col-12 .col-md-8</div>
  <div class="col-6 col-md-4">.col-6 .col-md-4</div>
</div>

<!-- Columns start at 50% wide on mobile and bump up to 33.3% wide on desktop -->
<div class="row">
  <div class="col-6 col-md-4">.col-6 .col-md-4</div>
  <div class="col-6 col-md-4">.col-6 .col-md-4</div>
  <div class="col-6 col-md-4">.col-6 .col-md-4</div>
</div>

<!-- Columns are always 50% wide, on mobile and desktop -->
<div class="row">
  <div class="col-6">.col-6</div>
  <div class="col-6">.col-6</div>
</div>

</div>
<div class="row m-3">
  <div class="col-8 col-lg-4 order-last">.col-8 .col-lg-4 .order-last</div>
  <div class="col-12 col-lg-6 order-first">.col-12 .col-lg-6 .order-first </div>
  <div class="col">.col</div>
</div>
...
</body>
</html>
```

Listing 4.2: An HTML code, illustrating an example of a Bootstrap layout.

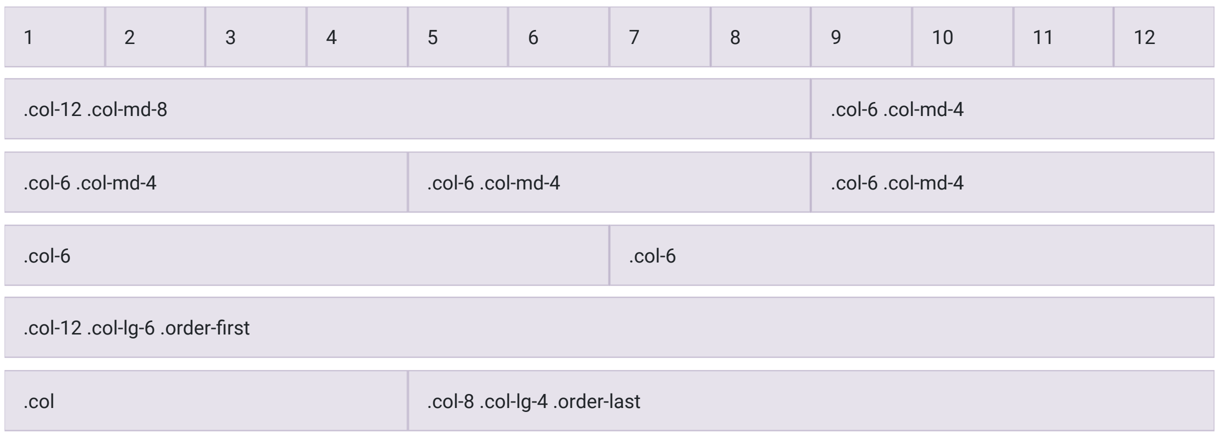


Figure 4.3: The rendered HTML page for devices with a screen width greater than 991 pixels.

4.1.10 i18Next

i18next[12] is a multi language software based on JavaScript. The term i18n in the computer world is associated with internationalization of software. The goal is to separate the software design with the translation process. Common i18n standards are:

- Plurals: Automatically selecting the right word (singular or plural). For example, 1 athlete and 4 athletes.
- Formats: Using the right date or currency format for a specific language.
- Context: Adding a context to translate correctly. For example, it can be useful to provide a gender to the translation to choose automatically between boyfriend and girlfriend.
- Interpolation: Used to add dynamic values to translation. The translation text keys are used as placeholders which are replaced with the passed values. An example is a welcome text for the user. Welcome back "User123", where "User123" is the passed value.

This project uses i18next to implement the basis for multilingual support and make it easier to extend for further translations without changing the software. Translations for languages are stored in separate JSON files and can easily be changed, even during run-time. i18next supports multiple languages and the translation files can be used by other platforms, such as Android, iOS, or NodeJS. The library also supports regional translations besides the base translation, for example, specific English translations for British English, Australian English or American English. Translations are mostly used by the front end, therefore a jQuery extension is used. This extension translates selected sections of the website without reloading the whole site. i18next also allows you to change the text of the website without changing anything in the HTML files (see Section 6.3.2).

4.1.11 Google Maps JavaScript API

Google Maps JavaScript API[13] is an online map service developed by Google. It creates an interactive map on the users' browser. By default, the road map also displays additional references, buildings, rivers, parks, and points of interest, such as restaurants, public transportation and shops. Google Maps is optimized for desktop devices, tablets, and mobile phones, which is essential for this project. An advantage is the good performance, which makes it possible to run it fluidly even on older devices. Google maps supports four different views types. By default, the view type "road map" is selected. Other view types include "terrain", which visualizes the elevation of the area, "satellite", which shows the map using aerial photographs, and "hybrid", a combination of the standard road map visualization and the satellite view. Google Maps is used to process geographical data and draw them on the map. Object overlays can only be added to the map using API calls. The following objects can be placed on the map:

- Single Location Overlays: Single location overlays are markers, icons, or images. These overlays are placed on the map using a single geographic coordinate. Overlays can also add text or custom animations.
- Poly-line: A poly-line is a shape displayed as a polygonal chain. The points of the poly-lines are geographical coordinates. Every point is connected to its successor with a line, which creates a path on the map.
- Symbols: Symbols are used to customize markers or poly-lines. Instead of using the default representation of a marker or poly-line, the overlays can be replaced with images or scalable vector graphics (SVG).

[12]i18next. `https://www.i18next.com/` (visited on 20/03/2019).
[13]Google, *Google Maps*. `https://maps.google.com/` (visited on 20/03/2019).

- Info Window: It is a pop-up window to display text, images, or HTML content in a pop-up bubble. The info window is assigned to a geographic location or an overlay on the map. To open the window, user interactions must be intercepted so that when a user clicks on a marker, the info window opens.

4.1.12 Google Charts JavaScript API

Google Charts JavaScript API[14] is a web service from Google to create interactive charts on web browsers. Charts are displayed as a scalable vector graphics (SVG) and they support a wide variety of types and customization. Section 4.5 discusses the components of a chart in more detail.

4.1.13 GPS Exchange Format (GPX)

The GPS exchange format (GPX)[15] is a file format with a standardized extensible markup language (XML) schema to store the global positioning system (GPS) coordinates. A GPX file can store three types of data: way-points (GPS coordinate of a location), routes, and tracks. A location point in the GPX file must store latitude and longitude. Optional information includes the elevation in meters and a creation timestamp. Event tracks are stored in a GPX file using the track type. A track consists of multiple ordered track points or way-points which describe the path of the track.

4.2 Database

The application uses an external MySQL database to access and store data. The database holds information of the events, athletes, GPS devices and coordinates. It is a central part for all software components.

4.2.1 Database Structure

The most important database tables are `Event`, `Devices`, `EventData`, and `EventHandover`.

4.2.1.1 Event

The `Event` table contains information about an event, such as name, description, and the start and end time. The event time is important for the front page to feature certain events that are either live or in the future. It is also used by the athlete tracking page to check if the event is still ongoing to avoid unnecessary update requests.

4.2.1.2 Devices

During an event, multiple devices are being used. These devices can be mobile phones, satellite tracking devices or conventional GPS trackers. The table also gives information about the used tacking devices and stores the last known location and timestamp. The location data always gets an update when a device sends one. Therefore every location update is stored in the `EventData` table.

4.2.1.3 EventData

The `EventData` table stores all data produced by the GPS devices. Every entry consist of the device ID, latitude, and longitude, a timestamp of when the device recorded the location and a timestamp of when the data was added to the database. Additional information includes the altitude, current speed and the compass direction in degrees, if a device can record this data.

[14]Google, *Google Charts*. `https://developers.google.com/chart/` (visited on 20/03/2019).
[15]TopoGrafix, *GPX: the GPS Exchange Format*. `https://www.topografix.com/gpx.asp` (visited on 20/03/2019).

4.2.1.4 EventHandover

In the `EventHandover` an athlete is assigned to a GPS device. It also stores information about athletes such as name, gender, or the skill level (beginner, pro, . . .). The event ID that assigns the athlete to an event and device ID that defines which devices the athlete uses is also part of the table. A so-called handover for GPS devices is possible, which means one GPS device can be used by multiple athletes during an event. Therefore, a start and end time must be stored to indicate how long a device is assigned to an athlete. This is beneficial if the number of available tracking devices is limited for an event. Reusing the stored athletes' data is not possible, therefore every event must create a new entry.

4.2.2 Database Access

Accessing data from the database is a key part of the application. Every time a user accesses the REST interface, the software has to make at least one database query. In addition to that, the server does not store any data permanently. Data received from the database is kept in a buffer, it gets processed and then it is sent to the client. The reason for that approach is that the data can change between requests and the result of one request can differ from another one, even if only seconds separate them. An important aspect for the software design is to use the separation of concerns software design principle. One of the most common design patterns used for this problem is the Data Access Object (DAO) pattern [Roman 2002]. This pattern separates the process of accessing the object from the database and sending the data to the client in three different parts:

1. Data Access Object Interface
2. Data Access Object concrete class
3. Model Object or Value

The Data Access Object Interface is a file that defines all operations that can be performed. This project uses Java Interface[16] as a solution. A Java Interface is similar to an abstract class or a header file in C++. The interface contains the signature of all methods without implementing the code. Comparing it with the real world, the template describes all the actions performed on the database. For example, retrieve all events, return all athletes participating in an event, or store a point of interest in the database. To execute a database query, the application has to establish a connection to the remote database. In Java, the so-called Java Database Connectivity API[17] is used to access the database. To establish the connection to the server URL and the database vendor, for example, MySQL, Oracle, or PostgreSQL, the username and password is needed. Switching between different vendors does not prevent performing SQL queries, as long it is supported by JDBC library[18]. The only requirement is that the database schema must be the same to avoid mapping problems.

The Data Access Object concrete class has to implement all methods defined by the Java Interface. This class has two responsibilities, handling SQL queries and mapping the records to the model object. These two task have to be done for every function call of the DAO concrete class. Every implemented method sends an SQL statement to the server using the JDBC, where it is executed. The received query result can have zero, one, or multiple records. A record set (row of database table) has one or multiple columns which holds the values of the database table. The second step is to map the query result to a model. The model object is called Plan Old Java Object, also known as POJO. A POJO is a simple Java class and only consists of a constructor, member variables, as well as getter and setter methods for all

[16]Oracle, *Interfaces and Inheritance*. `https://docs.oracle.com/javase/tutorial/java/IandI/createinterface.html` (visited on 20/03/2019).

[17]Oracle, *JDBC Overview*. `https://docs.oracle.com/cd/E19226-01/820-7688/gawms/index.html` (visited on 20/03/2019).

[18]Oracle, *Supported JDBC Drivers and Databases (Sun GlassFish Enterprise Server v3 Release Notes)*. `https://docs.oracle.com/cd/E19226-01/820-7688/gawms/index.html` (visited on 20/03/2019).

variables. Every record is mapped to an object where every column is stored in a member variable of the object, using the setter methods. Depending on the SQL query, the DAO method returns a POJO, a list of POJOs or a simple data type. This project uses three DAOs to map database data to java objects:

- In the `AthleteDAO` class is responsible to receive and map the data from the `EventHandover` table.
- The `EventDAO` class handles all database queries that are related to the live events.
- The `EventDataDAO` class handles all queries made on the `EventHandover` to receive coordinates of athletes.

Accessing the athletes' coordinates is one of the most important interfaces. During a live event, the client requests the location data of all athletes in an interval of a few seconds. Furthermore, the location data of every athlete is necessary to calculate several statistics described in Section 4.3, which are requests by the client during the event. With an increasing number of athletes and clients, the number of queries can increase significantly. Depending on GPS tracker, the position update varies. Therefore, multiple requests can produce the same result during a certain period of time. To avoid unnecessary database queries and reduce the server load, the `EventHandoverDAO` has implanted a caching mechanism. Before requesting the whole list of geographic coordinates for an athlete from the server, the class checks if the stored locations in the cache are still valid. If there are no newer data records in the database, the algorithm returns the data from the cache. If the cache is not valid anymore, the server makes a request. The cache is only used during a live event and for that reason a routine clears the cache every few hours.

4.2.3 Implementation

This section explains the implementation of the most important functions of every Data Access Object (DAO) and shows the structure of each return value. As mentioned above, every DAO maps the SQL records to the corresponding POJO.

4.2.3.1 Event Data Access Object

The Event Data Access Object returns either a `Event` object or a list (`List<EventData>`) of events. The `EventDAO` contains following methods:

- `Event getEventById(int id)`:
  Description: Returns the event with the given ID.
  Arguments: `id` is the even ID.
  SQL Query: `SELECT * FROM Event WHERE idEvent = id`

- `List<EventData> findall()`:
  Description: Returns a list of all events.
  Arguments: -
  SQL Query: `SELECT * FROM Event`

- `Event findEventByHandoverId(int idHandover)`:
  Description: To get the event for an athlete, two queries are executed. The first query finds the event ID based on the athlete ID. Then it uses the event ID to return the event data.
  Arguments: `idHandover` is the athlete ID.
  SQL Query: `SELECT * FROM Event WHERE idEvent = id` returns the event ID. Then use function `getEventById(int id)` to get the event.

- `List<Event> findEventFromTo(long from, long to, int limit, int offset)`:
  Description: The method returns a list of events which are in a certain time frame.
  Arguments: The values `from` and `end` define the start and end point of the time frame. `offset` defines the index of the first record and `limit` the number of list elements. Both variables are used for the

pagination.
SQL Query: "SELECT * FROM Event WHERE dateEventStart BETWEEN from AND to ORDER BY dateEventStart DESC LIMIT limit OFFSET offset

4.2.3.2 Athlete Data Access Object

The Athlete Data Access Object returns either a Athlete object or a list (List<Athlete>) of athletes. The AthleteDAO contains the following methods:

- Athlete findAthleteByHandoverId(int idHandover):
  Description: Returns the athlete with given ID.
  Arguments: `idHandover` is the ID of the athlete.
  SQL Query: SELECT * FROM EventHandover WHERE idHandover = id

- List<Athlete> findall():
  Description: Returns a list of all athletes.
  Arguments: -
  SQL Query: SELECT * FROM EventHandover

- List<Athlete> findAthletesByHandoverId(List<Integer> ids):
  Description: Returns a list athlete with the given IDs.
  Arguments: `ids` is the list of all athlete IDs that should be returned.
  SQL Query: SELECT * FROM EventHandover WHERE idHandover IN ids.

- List<Athlete> findAthletesFromEvent(int idEvent):
  Description: Returns a list of all athletes that take part in a event with the given ID.
  Arguments: `idEvent` is the ID of event.
  SQL Query: SELECT * FROM EventHandover WHERE idEvent = idEvent

4.2.3.3 Athlete Coordinates Data Access Object

The DAO that accesses the athlete coordinates is called EventDataDAO. Every function of the class returns a list of coordinates (List<EventDataDAO). The EventData class which contains the GPS data contains following methods:

- List<EventData> getCoordinatesForAthlete(int athleteId)):
  Description: Returns the list of coordinates for a given athlete. The athlete ID is used to obtain how long the athlete uses a tracking device.
  Arguments: `athleteId` is the ID of the athlete.
  SQL Query: SELECT * FROM EventData WHERE deviceID = deviceId AND timestamp BETWEEN startTimeStam p AND endTimeStamp ORDER BY timestamp ASC

- List<EventData> findall():
  Description: Returns all stored coordinates of all athletes.
  Arguments: -
  SQL Query: SELECT * FROM EventData

- List<EventData> findEventDataByDeviceId(String id):
  Description: Returns all coordinates of a tracking device with the given ID.
  Arguments: `id` is the ID of tracking device.
  SQL Query: SELECT * FROM EventData WHERE deviceID = id ORDER BY eventdata.timestamp ASC

4.3 Athlete Statistics

Another important part besides the athlete visualizations is to provide information about the performance of athletes. The users should be able to compare athletes during an event based on statistical information. The goal is to design a software module that is easy to extend. Developers who want to add other statistical evaluations for an athlete should not need to struggle with other functionalities of the framework. The idea is that developers just create a new Statistics class and implement the algorithm which returns the result. It should not be necessary to make changes in the user interface or extend the REST interface when adding a new statistical calculation.

To master these specifications the software design uses the following approach. Each statistic class must extend a method which includes the algorithm. The only limitation of the implementation is that the calculated value must be a `Double` data type. The return units of current implemented statistics are meters, kilometers per hour, minutes per kilometers or a time span measured in seconds. All these values can be stored as `Double` data type. Using a Java `String` as a return type could lead to the following problems. For example, the calculation of how long an athlete is on the track returns a well format string in hours, minutes, and seconds instead of a numerical value. For example, the calculation of the time an athlete spends on the track returns the format string in hours, minutes, and seconds instead of a numerical value.

If a client requests the duration time and receives a formatted string, this limits further use of the data. Adding the string to the user interface is easier, but it is not possible to perform any further calculations. In addition to that, the string may not be readable for all users due to a language barrier. Sticking to the separation of concerns in design principle, the decision is to let the client who requests the data from the REST interface deal with the representation or formatting.
The data to create statistical evaluations is limited, but what is available is the athletes' coordinates, ordered by creation date, which can be accessed using the DAO class. In addition to that, the statistic class stores the event ID to access the coordinates of the racetrack, if available. The track files contain a list of geographic coordinates and the altitude of every location. Every implementation of a `Statistic` class uses a unique `Enum`. An `Enum` is a class which represents a group of constant variables. The constant variables are the implemented statistics. The `Enum` class of the statistics is used by the REST interface to display all implemented statistics, and by the factory class which has the task to create a statistic object based on the unique name.

4.3.1 Distance

Calculating the distance between two geographic coordinates is an important operation for multiple algorithms. This problem is solved using Vincenty's formulae [Vincenty 1975] which requires a World Geodetic System.

4.3.1.1 World Geodetic System

The World Geodetic System (WGS) is a standard used in geodesy, for geographical navigation and GPS coordinates. The introduction of a standardized system started a breakthrough in international surveying and was used as a new standard for many survey systems. The first WGS was introduced in 1960 and named WGS 60 and the improved version WGS 66 [WGS Implementation Committee 1967] was released a few years later in 1967. This system allows to measure the distance between two points. A point in the grid system consists of latitude, longitude, and elevation. Throughout the years, the system was improved due to the development of new technologies which improved the satellite systems and accuracy of coordinates improved as well. A new geodetic system was introduced between 1971 and 1972, called WGS 72 [Seppelin 1974]. More and more applications required an accurate system to navigate and to extend the geographic coverage. WGS 72 was too inaccurate for many applications, so a new system was developed. The new system called WGS 84 [National Imagery and Mapping Agency 2000] was developed during 1980 and established in 1984. The systems introduced the global positioning system

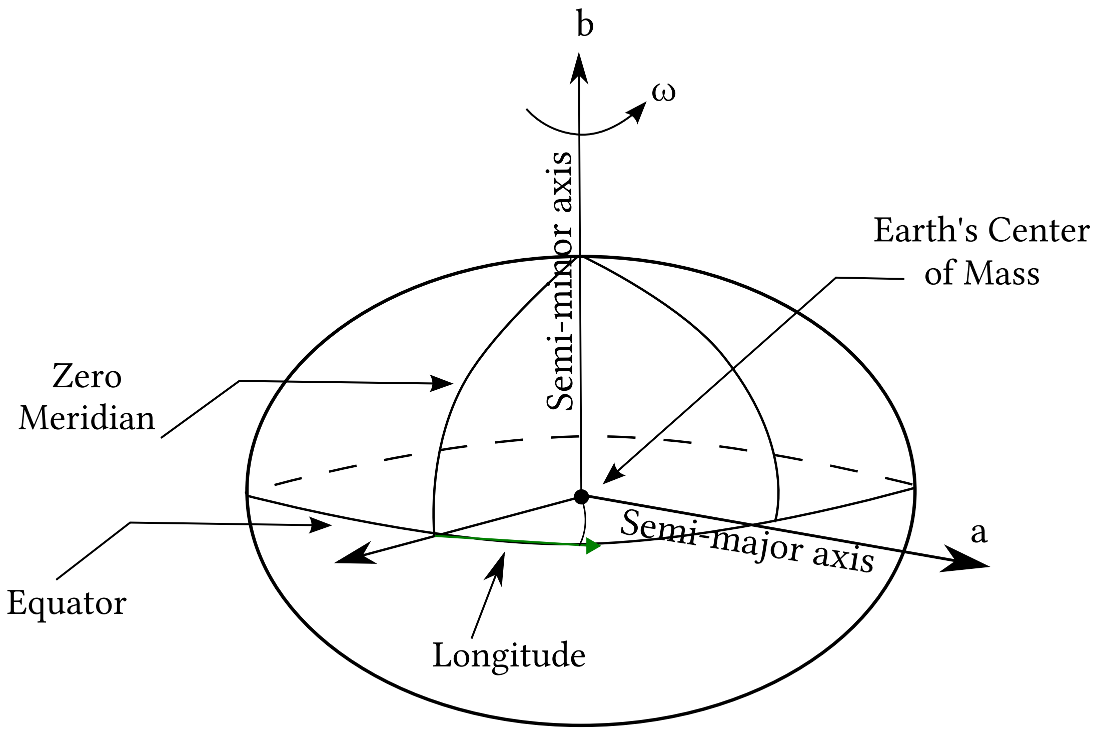


Figure 4.4: An example of the reference model of the WGS 84 ellipsoid. [Image adopted by the author of this thesis from WGS Implementation Committee [1967].]

(GPS) as a new reference system and improved over the years. The last update was in 2004 and is the current WGS.

The WGS 84 System consist of following parts:

- Reference ellipsoid: It represents the surface of the earth and it is possible to localize points of interest using latitudes and longitudes.
- Geoid: As geoid in WGS 84, the GM96 geoid [Lemoine et al. 1998] is used. It is a more complex representation model of the earth. It includes the gravitational force of the earth, a more detailed representation of earth surface and helps to define the altitude.
- Points of fundamental stations: It contains a list of three-dimensional points that, when aligned, define the model of the earth. For the calculation of the distance between two GPS coordinates, the reference model of ellipsoid is required. The earth ellipsoid defined by WGS 84 contains following parameters:
  - a represents the equatorial radius with the value of 6378137 meters. It is called semi-major axis.
  - The flattening of the sphere is represented by the variable f. It is calculated with $= 1/298.257223563$.
  - The inverse flattering $1/f$ is 298.257223563.
  - The polar radius b is calculated with the help of a and f. $b = a(1 - f)$.

Figure 4.4 shows an example of the WGS 84 ellipsoid.

4.3.1.2 Vincenty's Formulae

For the calculation of the distance between two GPS coordinates, the algorithm of Vincenty's formulae [Vincenty 1975] is used.

Notation of parameters:
a, b are the minor or major semi axes of the ellipsoid.
f is the flattening of the ellipsoid.
ϕ_1, ϕ_2 are the latitudes of the start and the end coordinate.
$\lambda_1 \lambda_2$ are the longitudes of the start and the end coordinate.
$L = L_1 - L_2$ is the difference in the longitude of the start and end point.
α_1, α_2 are forward azimuths at the points.
α is the azimuth at the equator, along the semi major axis.
s is the ellipsoidal distance between the start and the end point.
σ is the arc length between the two points on the auxiliary sphere.
$U_1 = \arctan((1-f)\tan\phi_1)$ and $U_2 = \arctan((1-f)\tan\phi_2)$, are the latitude on the auxiliary sphere, called reduced latitude.

Start of the Inverse Formula

$$\lambda = L \tag{4.1}$$

$$\sin\sigma = \sqrt{(\cos U_2 \sin\lambda)^2 + (\cos U_1 \sin U_2 - \sin U_1 \cos U_2 \cos U_2 \cos\lambda)^2} \tag{4.2}$$

$$\cos\sigma = \sin U_1 \sin U_2 + \cos U_1 \cos U_2 \cos\lambda \tag{4.3}$$

$$\sigma = \arctan\frac{\sin\sigma}{\cos\sigma} \tag{4.4}$$

$$\sin\alpha = \frac{\cos U_1 \cos U_2 \sin\lambda}{\sin\sigma} \tag{4.5}$$

$$cos^2\alpha = 1 - \sin^2\alpha \tag{4.6}$$

$$\cos(2\sigma_m) = \cos\sigma - \frac{2\sin U_1 \sin U_2}{\cos^2\alpha} \tag{4.7}$$

$$C = \frac{f}{16} * \cos^2\alpha * (4 + f * (4 - 3 * \cos^2\alpha)) \tag{4.8}$$

$$\lambda = L + (1-C)f * \sin\alpha(\sigma + C\sin\sigma(\cos(2\sigma_m) + C\cos\sigma(-1 + 2cos^2(2\sigma_m)))) \tag{4.9}$$

The iterative algorithm starts with (4.2) and ends with (4.9). After (4.9) the new λ is compared with the old value L, shown in equation (4.1). The iteration stops after a defined amount of loops, or λ converge. λ converges if the difference between the new λ of the iteration step and the initial $\lambda = L$ is smaller than 10^{-12}. However, the indirect algorithm may not converge or slowly converges if both coordinates are nearly antipodal points. In this project, the possibility that both GPS coordinates are near the poles is practically zero. The implemented algorithm uses a maximum of 20 iterations. Thaddeus Vincenty states in his paper that standard points take about two to four iterations in most cases. In his tests for antipodal points, the algorithm needs about 18 iterations to find a result.

After the iteration step, the distance s is calculated:

$$u^2 = \cos^2 \alpha \frac{a^2 - b^2}{b2} \tag{4.10}$$

$$A = 1.0 + \frac{u^2}{16384} * (4096 + u^2(-768 + u^2 * (320 - 175 * u^2))) \tag{4.11}$$

$$B = \frac{u^2}{1024}(256 + u^2 * (-128 + u^2 * (74 - 47 * u^2))) \tag{4.12}$$

$$\begin{aligned}\Delta\sigma &= B * \sin\sigma(\cos(2\sigma_m) + \frac{B}{4}(\cos\sigma(-1 + 2\cos^2(2\sigma_m)) - \\ &\frac{B}{6}\cos(2\sigma_m)(-3 + 4\sin^2\sigma)(-3 + 4\cos^2(2\sigma_m))))\end{aligned} \tag{4.13}$$

$$s = bA(\sigma - \Delta\sigma) \tag{4.14}$$

4.3.1.3 Vincenty's Formulae Extended Calculation

Vincenty's inverse formulae are a popular approach for calculating the distance between two coordinates. The algorithm is very accurate and fast in comparison to others. However, it is possible to extend this algorithm. The inverse formulae do not consider the difference in height between two coordinates. Most GPS trackers also store the altitude for every coordinate, which is mandatory for the modification.

The Notation:
e_1, e_2 the elevation of the start and end coordinate.
ϕ_1, ϕ_2 the latitude of both coordinates.
a the semi major axis defined by WGS 84
f flattening of the ellipsoid defined by WGS 84.
s is the calculated distance of the two coordinates using Vincenty's inverse formulae.

Algorithm

$$e_{\text{mean}} = \frac{(e_1 + e_2)}{2} \tag{4.15}$$

$$\phi_{\text{mean}} = \frac{(\phi_1 + \phi_2)}{2} \tag{4.16}$$

$$a_1 = a + e_{\text{mean}}(1 + f + \sin\phi_{\text{mean}}) \tag{4.17}$$

$$b_1 = (1.0 - f)a_1 \tag{4.18}$$

$$s_1 = \sqrt{s^2 + (e_2 - e_1)^2} \tag{4.19}$$

Based on the new semi major axis, calculated in (4.17), a new ellipsoid is calculated. The final distance s_1 between both three-dimensional coordinates, consisting of latitude, longitude, and altitude, is calculated in (4.19).

The athlete track consists of multiple GPS points. The distance of a track is calculated with iteration over all points. In each iteration step, the algorithm calculates the distance between the current point and its successor using Vincenty's formula with a modification. The distance is added to the cumulative distance of the previous points. This is done for every GPS point in the list, which results in the total distance of an athlete.

4.3.1.4 Evaluation

The algorithm consists of multiple loops. The first loop adds up all the distances between the GPS points of an athlete. The second loop is in Vincenty's indirect formula, which can be repeated up to 20 times in the worst case. In addition to that, this must be done for all athletes on the track at the same time. Considering all these loops and amounts of GPS coordinates, there were doubts as to if the algorithm would perform well. The implementation was tested with some GPX files and it performed quite well:

- 12 milliseconds for 347 coordinates created form the author of this thesis[19].
- 3 milliseconds 370 coordinates created from FH-Kärnten[20].
- 61 milliseconds 4035 coordinates created from FH-Kärnten[21].
- 59 milliseconds 6926 coordinates created from FH-Kärnten[22].
- 24 milliseconds 5769 coordinates created from FH-Kärnten[23].

4.3.2 Duration

There are two possibilities to determine how long an athlete is on the track. The first option is to use the GPS records of an athlete using the timestamps, the first and last created GPS coordinates and calculating the difference to get the duration. The second possibility is to take the start time of an event into consideration. The start timestamp is either the start time of the event or the timestamp of the first created coordinate, depending on which one is newer. In this case, the duration is the difference between the newer timestamp and the timestamp of the last entry from the GPS device. Using the second approach could lead to inaccuracies in other calculations. For example, calculating the distance of an athlete uses all coordinates, but calculating the duration would not. This would make all calculations that lean on the distance or duration inconsistent, because the statistics would use different sets of data for the same athlete. As observed in multiple instances, the event starting time is more like a reference value. It is possible for events to start before or more likely after the predefined time. Taking the problems of the second option into account, the decision was to implement the first approach.

4.3.3 Speed

The speed statistics are a snapshot of the current performance of an athlete. It only takes the two last created coordinates in account and calculates the speed in kilometers per hour, as described in Average Speed Section 4.3.4. To extend this measurement, one can't only consider the last two created points. Increasing the measuring window of the coordinates could lead to a better performance evaluation over a short time span. Selecting the right amount of coordinates is crucial in order to not falsify the result and make it comparable with other athletes. Choosing a time window of a predefined amount of seconds or distance is necessary to filter out the coordinates that are relevant for the calculation. While analyzing existing GPS records, this approach can be difficult. Depending on the used GPS device, the update rate for sending coordinates to the server varies and connection problems of GPS trackers can occur. If the GPS tracker records a location, the device might not send the update immediately to the server, which means records over a certain time span or distance are not always available. An interpolation function is required to generate the missing GPS coordinates to solve this problem. This approach is too complex to implement and therefore not suitable.

[19]Stary, *Simulation GPX File*. `https://www.wikiloc.com/outdoor-trails/simulation-track-35326911` (visited on 20/03/2019).

[20]Fachhochschule Kärnten, *Kosiak - Athlete 336*. `https://www.wikiloc.com/trail-running-trails/fh-karnten-35326765` (visited on 20/03/2019).

[21]Fachhochschule Kärnten, *Ice2Ice - Athlete 300*. `https://www.wikiloc.com/cycling-trails/ice2ice-35327036` (visited on 20/03/2019).

[22]Fachhochschule Kärnten, *Ice2Ice - Athlete 301*. `https://www.wikiloc.com/cycling-trails/ice2ice-athlete-301-35327087` (visited on 20/03/2019).

[23]Fachhochschule Kärnten, *Ice2Ice - Athlete 302*. `https://www.wikiloc.com/cycling-trails/ice2ice-athlete-302-35327132` (visited on 20/03/2019).

4.3.4 Average Speed

The average speed should make it easier for the user to compare the performance of athletes over the whole event. Calculating the speed of an athlete uses the previously described statistics. The distance and the elapsed time is needed, with the distance converted to kilometers and the time to hours. The speed in kilometers per hour is calculated as speed = distance/duration. The value duration shows how long the athlete is active (see Section 4.3.2) in hours and distance is the covered distance of the GPS records in kilometers (see Section 4.3.1).

4.3.5 Pace

The pace is a widely used measurement in sports to analyze the performance of athletes. The pace defines how much time an athlete takes to cover a certain distance. In all of sports, the unit used is minutes per kilometer or miles. In other words, how many minutes an athlete takes to cover one kilometer. The formula is pace = duration/distance. The duration is how long the athlete is active in minutes (see Section 4.3.2) and distance is the covered distance of the GPS records in kilometers (see Section 4.3.1).

4.3.6 Arrival

The arrival gives an approximate value of how many seconds an athlete needs to finish the race. This calculation requires a GPX file to calculate the distance of the racetrack to determine how many kilometers an athlete has to cover. If a GPX file is not available, another option is simply to set the distance of the racetrack in a property file, as described in Section 4.6. The arrival is calculated arrival = (distance(RaceTrack) − distance(Athlete))/avgSpeed(Athlete), where RaceTrack contains the GPS coordinates of the GPX file and Athlete contains the GPS coordinates of the athlete. The function distance calculates the distance of the GPS coordinates in meters (see Section 4.3.1) and avgSpeed calculates the speed of the GPS records in meters per second (see Section 4.3.4).

A problem with GPS trackers is that the recorded coordinates are not 100 percent accurate. More often than not, the GPS tracker produces outliers. This means that if an athlete is running on the racetrack, the recorded GPS coordinates may not be on the track and the calculated distance of GPS coordinates may differ from the actual distance. During the live tests, it was possible to monitor the inaccuracy of the GPS records increasing or reducing the covered distance of an athlete. This variance also leads to a small inaccuracy of the calculation. Another problem occurs if the event consists of different legs, for example, a running and a cycling leg. The speed differential makes the average speed inaccurate because it is calculated over all coordinates. Depending on the duration of the running and cycling leg, the arrival time can either be too early or too late.

There are many ways to improve the prediction of the arrival time, regarding accuracy. For example, instead of using the basic distance calculation (see Section 4.3.1), the distance can be projected using the algorithm discussed in Section 4.4.4. This reduces miscalculations of the distance due to inaccurate GPS coordinates. Furthermore, it would be possible to improve the prediction by considering the athlete's skills. Based on the capabilities of an athlete, it would be easier to predict whether the speed will remain the same during the course of an event or not. In addition to that, it would be possible to create a profile of athletes as discussed by Smyth and Cunningham [2017], and Berndsen et al. [2017] by storing the running behavior or the performance based on certain characteristics of a racetrack, for instance. These profiles could be used to filter out athletes with similar skills who attend the same event. Based on the live or previous measurements of the athletes, it would be possible to predict the performance of an athlete for the current event which can improve the correctness of the arrival time.

GPS Error Sources	Error in Meters
Satellite Clocks	2.1
Ephemerides Data	2.1
Ionosphere	4.0
Troposphere	0.4
Receiver Noise	0.5
Multipath	1.4

Table 4.2: The error sources and the typical error in meters caused by inaccurate positioning of GPS devices [Jansen et al. 2016].

4.4 Athlete Mapping

Recorded locations of GPS devices may not always be accurate. To calculate the position of a device, it needs the signal of at least three satellites. Due to some obstacles like mountains, trees, or buildings, the position becomes more inaccurate as the signal bounces between those objects; this problem is called multipath error. A Weather front can also cause inaccurate positioning due to disturbances in the troposphere [Gregorius and Blewitt 1998]. Other factors are the satellite clock, ephemerides data, ionosphere, and receiver noise [Ranacher et al. 2015; Drawil et al. 2013]. Table 4.2 shows the source of error and the typical error in meters [Jansen et al. 2016].

As a result of these errors, the path of an athlete may not always be accurate. The path of the GPS records does not always match the real path. This leads to the issue that the path of an athlete does not align on the racetrack or predefined routes of the event. To improve the visualization of athletes' tracks, several mapping algorithms are implemented.

4.4.1 Google Map to Road API

Google Map to Road API[24], is an interface provided by Google and tries to map recorded data from the GPS devices to the road. To use this interface, it is recommended to send as many GPS coordinates at once to the server. In this way the algorithm can determine faulty GPS records more easily because it is simpler to predict the moving direction. It is also suggested that the distance between points are not too far apart. The reason for that is, if two GPS records are too far away from each other, it becomes harder to predict the path and map the points to the road. The maximum amount of positions the API can handle is limited to 100 locations per request. If possible, every API request includes 100 positions. This helps to accomplish a better mapping, and reduces the amount of API calls. Google charges every API request made, using the Google Road API. To perform a request, an API key is needed, which can make up around 20.000 requests without getting charged[25]. If the athlete track consists of more than 100 GPS records, the data must be split up and make multiple requests. This can lead to the mapping algorithm not mapping the points to the right road if the coordinates of the previous request are not included. If the data must be split into multiple fragments, the pagination uses an overlap window. Every fragmented list contains positions of the previous one. for example, the last five elements of the second fragment, used as the first five elements of the third fragment. In the implementation, the fragment for a request always starts with five coordinates of the previous fragment. The Google Road to Map API works great for events where the track runs along marked roads, however, running events like mountain runs or cross-country runs are not suited for the mapping algorithm. In most cases, these events use hiking trails or non marked roads as racetracks. These paths are not part of the Google Maps route system and therefore they are not

[24]Google, *Roads API*. `https://developers.google.com/maps/documentation/roads` (visited on 20/03/2019).

[25]Google, *Roads API Usage and Billing*. `https://developers.google.com/maps/documentation/roads/usage-and-billing` (visited on 20/03/2019).

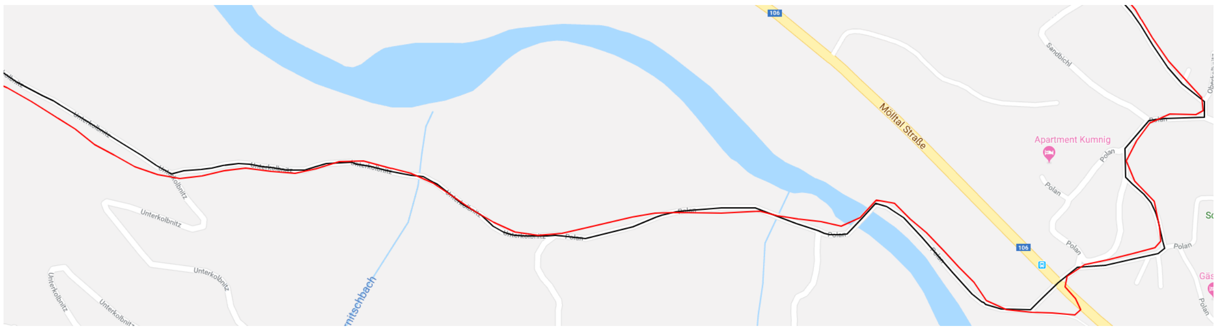


Figure 4.5: The red track is the path using the original GPS coordinates and the black track is the mapped path using the Google Map to Road API and the interpolation option.

usable for mapping the GPS data. Due to this, the mapping algorithm may not return any mapped data points for a request. If the API does not return any valid coordinates, the raw GPS locations of an athlete are used instead.

Another functionality used is the option to interpolate the GPS points. This Feature improves the alignment of the GPS data to a road. In order for this to be accomplished, the algorithm adds multiple points between the recorded coordinates. The added location points follow the mapped road and create a naturally shaped path along the road. Figure 4.5 shows the difference between using interpolate coordinates and the original GPS coordinates. Adding those interpolate points also increases the data size of the record set. A GPX file[26] containing 347 GPS points increases to 774 entries using the Google Map to Road API with interpolation option enabled.

The location points almost double which also depends on the geometric characteristic of a road. Therefore, using this mapping algorithm may increase the data usage by 200%. Another downside to using the mapping algorithm is that the new GPS records are not usable for statistical calculations, discussed in Section 4.3. The mapped coordination does not include information on the creation date or elevation. It is possible to restore this information for non interpolate coordinates, but not for interpolate ones. Restoring the data for interpolated coordinates would be too complex and time-consuming.

4.4.2 Google Direction API

The Google Direction API[27] is another web-service provided by Google. This service is normally used to create and calculate a route between two points. Typically, a request consists of an origin and a destination point. The application sends a request including this information using the Google Direction API. The response of the server consists of two parts. The first part stores information about the locations points sent to the server as follows:

1. A Status of the created route. OK indicates that it was possible to create a valid route between origin and destination. ZERO_RESULTS means that the algorithm was not able to find a route. This error also indicates a possible error regarding a faulty location.

2. The response includes an indicator about the matching of the request. If the indicator partial_match is true, it means the location of the origin or destination were not able to match and were re-positioned to create a route.

[26]Stary, *Simulation GPX File*. `https://www.wikiloc.com/outdoor-trails/simulation-track-35326911` (visited on 20/03/2019).

[27]Google, *Directions API*. `https://developers.google.com/maps/documentation/directions` (visited on 20/03/2019).

The second part consists of multiple route elements. The API can provide multiple route options to improve the route or decrease the traveling time. A route element can contain the following information:

1. A text that describes the route.

2. The route may consist of multiple legs. A leg is the route between two way-points within the route. It has its own origin and destination, and it includes information about the distance of the leg and the way-points that create the route.

3. If the algorithm reorders the way-points to improve the route, it will show the reordered locations in the request. Improving the route means a decrease in distance or travel time.

4. The response also includes an encoded poly-line of the route. This poly-line connects the single legs of the route and creates a path on the map. This path is aligned and smoothed to the shape of the route. The received poly-line is encoded using a compression algorithm[28].

The Google Direction API is not necessarily meant to be used as a mapping algorithm, but it is possible to do so. To achieve this, the implementation uses the following approach. Each request consists of an origin and destination location, and one may also add way-points. By adding way-points to the request, the requested route must add these locations into the route calculation. A request can have a maximum of 25 location points. Two locations points for origin and destination and an additional 23 positions for the way-points. Therefore, the GPS records of an athlete or a racetrack must be divided into fragments. Each fragment holds at least two elements and 25 locations at most. If the amount of recorded locations is smaller than the limit of the request, the procedure only creates one fragment. In this case the first element is the origin location and the last one is the destination point. The other coordinates are the way-points of the request. If the number of elements is greater than 25, there are at least two fragments. The first fragment includes the first 25 coordinates of the GPS records, and for the second and following fragments the first element is the last element of the previous fragment. In other words, the destination point of the previous fragment is the origin location of the current fragment. It is valid for all fragments that the first element is always the origin, the last element the is destination and the elements in between are the way-points for the request. Using only two GPS coordinates for a request, one location as the origin and the second as the destination is not as beneficial because it would increase the number of API calls and it would take significantly more time to map the locations. In comparison, 25 GPS records only takes one API call using the implemented fragmentation and 24 API calls without using way-points.

To improve the mapping using Google Direction API, some optional parameters are used:

- The travel mode indicates which type of roads are usable to generate the direction. The usable transportation modes are driving, walking, and bicycling. Using the walking mode is the best option because it is not limited to roads but can also include roads that are used for driving cars.

- The option to create alternative routes is disabled to reduce the response time.

- Because the application is used by outdoor events, mainly running, swimming, or bicycling, the option to add indoor routes into the route creation is disabled.

- By default, the web service tries to improve the route by reordering the way-points so that the distance or the estimated time to walk or drive the route is reduced. This is not ideal as reordering the way-points falsifies the original track of an athlete, therefore it is disabled.

Every fragment makes a request using these parameters. The response of the Google Direction API includes a encoded poly-line, as for example, the geographical coordinates of Inffeldgasse 25, with a

[28]Google, *Encoded Polyline Algorithm Format*. `https://developers.google.com/maps/documentation/utilities/polylinealgorithm` (visited on 20/03/2019).

```
function decode(encodedString)
index = lat = lng = 0

while (index < encodedString.length)
  shift  = 0
  result = 1

  do
    char = encodedString[index]
    index++
    a = getAsciiCode(char) - 63 - 1;
    result += a << shift;
    shift  += 5
  while a >= 31

  if result & 1 != 0
    lat += ~(result >> 1)
  else
    lat += result >> 1

  do
    char = encodedString[index]
    index++
    a = getAsciiCode(char) - 63 - 1;
    result += a << shift;
    shift  += 5
  while a >= 31

  if result & 1 != 0
    lng += ~(result >> 1)
  else
    lng += result >> 1

  coordinates.add((lat*1e-5, lng * 1e-5))
do

return coordinates
```

Listing 4.3: A pseudo code of the decoding functions to decode the encoded poly-line of a Google Directions response.

47.05828 latitude and 15,46032 longitude, represented by the encoded string "gaf G_rj}7A". To decode an encoding string, the algorithm shown in Listing 4.3 is needed.
An example of how to decode the encoded String "gaf G_rj}A" is shown in Table 4.3 and Table 4.4.

To summarize the steps, each fragment makes a request using the Google Direction API. The next step is to decode the smoothed poly-line from the response. After that, the decoded coordinates are added to the result list. It is important to keep the elements in the original order so that the mapped coordinates are not mixed-up. Every fragment must repeat these steps.

Using the mapping does not provide a link between the original locations and the new mapped point. Therefore, it is not possible to restore the altitude or timestamp of a mapped point unlike using the Google Roads API. For that reason, the mapped coordinates are not suitable for statistical calculations

Input	Variables					Shift	Operation
Character	ASCII Code	Index	result	b	shift	B <<shift	B >= 32
			1,00	0	0		
g	103	0	40,00	39	0	39	continue
a	97	1	1.096,00	33	5	1056	continue
f	102	2	40.008,00	38	10	38912	continue
~	126	3	2.071.624,00	62	15	2031616	continue
G	71	4	9.411.656,00	7	20	7340032	break
Calculating the latitude	Condition: result & 1 != 0 =>false latitude = (result >>1) * 0,00001 = 4705828 * 0,00001 = 47,05828						

Table 4.3: The steps to decode the latitude.

Input	Variables					Shift	Operation
Character	ASCII Code	Index	result	b	shift	B <<shift	B >= 32
			1		-5		
_	95	0	32	31	0	31	break
r	114	1	1.632	50	5	1.600	continue
j	106	2	44.640	42	10	43.008	continue
}	125	3	2.043.488	61	15	1.998.848	continue
A	65	4	3.092.064	1	20	1.048.576	Break
Calculating the longitude	Condition: result & 1 != 0 =>false longitude = (result >>1) * 0,00001 = 3.092.064 * 0,00001 = 15,46032						

Table 4.4: The steps to decode the longitude.

as discussed in Section 4.3. The mapping algorithm can be used for athletes or GPX files of an event. Comparing the number of elements between the original coordinates and the mapped ones, one can see it varies significantly. Depending on the number of created coordinates and the geometrical shape of the route, the amount of additional locations can be the same or many times higher. In Figure 4.6 for example, 32 GPS coordinates, represented by the red markers, are used as input for the mapping algorithm. The mapped poly-line consists of 162 location points. This is five times more, than the original number of elements. Applying the mapping algorithm to the GPX file[29] used in Section 4.4.1, the number of way-points reduces from 347 to 345 coordinates.

Comparing this algorithm to the one described in Section 4.4.1, it provides slightly better mapping. The reason for this is that this mapping approach also includes sidewalks and cycling paths. Most of the events use these particular routes and the Google Direction API also includes them as a possible target to map the original location points.

[29]Stary, *Simulation GPX File.* https://www.wikiloc.com/outdoor-trails/simulation-track-35326911 (visited on 20/03/2019).

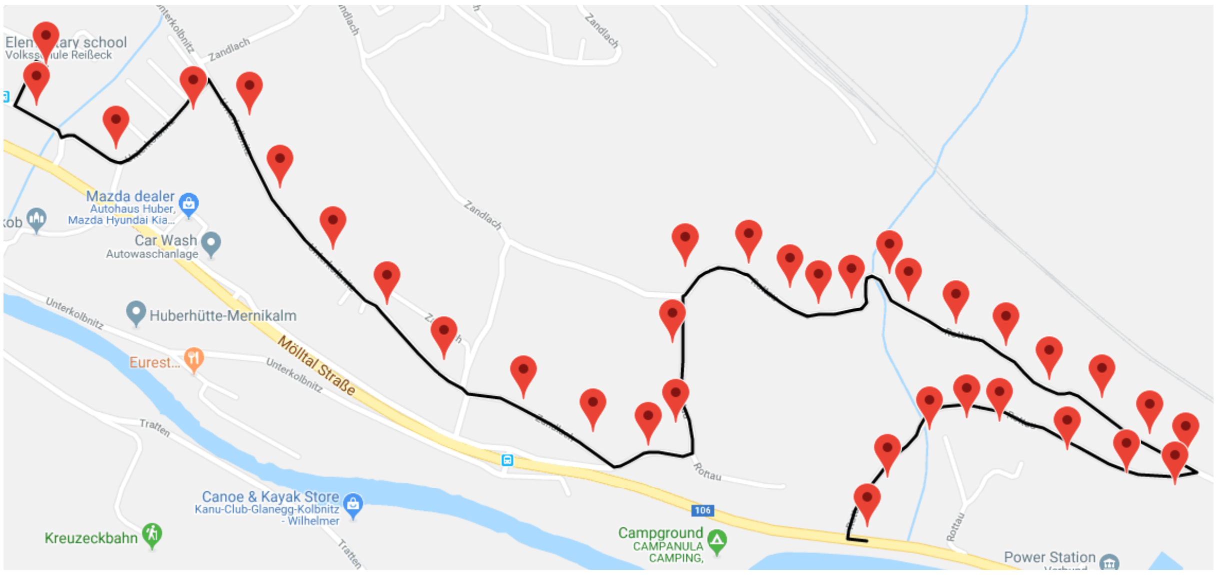


Figure 4.6: The red markers are the GPS coordinates of device and the blue line is the align path using Google Direction API.

4.4.3 Map on Track

The Map on Track function maps an athlete based on distance covered. This approach uses the coordinates of the event track as target mapping. The first step is to find two way-points on the racetrack based on the distance of an athlete
as shown in Figure 4.7.

The algorithm shown in Listings 4.4 takes two arguments. `racetrackCoordinates` are the locations from the GPX file and `athleteDistance` is the current distance of the athlete. First, the algorithm iterates over all coordinates of the racetrack and adds up the distance, using the method described in Section 4.3.1. The iteration stops if the summed up distance is greater than the covered distance of the athlete. Therefore, the new mapping point is between the previous two coordinates of the current iteration. With these two points it is possible to calculate the distance ratio and get the mapped point (Line 11). The ratio requires the distance between the two points (`point1` and `point2`) and the distance between point `point1` and the projected point of the athlete (Line 10). To calculate the distance between the points, the helper variables `distance` and `previousDistance` are needed, as defined in Line 2. Subtracting the summed up distance from the previous iteration of the distance of the current iteration results in the distance between these points (see Line 9). The distance between `point1` and the projected point is the difference of the `previousDistance` and the `athleteDistance`, as shown in Line 10). The distance ratio is the distance between the first point and the mapped point, divided by the total distance (see Line 11). The next step is to calculate the difference of the latitude and longitude between the `point1` and `point2`, as shown in Lines 13 and 14. Multiplying the difference with the distance ratio gives us the difference in latitude and longitude between `point1` and the new mapped coordinate `pointM` in decimal degrees. The mapped athlete track includes all elements of the racetrack that are a predecessor of `point2` and the new mapped point as the last element, shown in Lines 18 and 19.

Calculating the mapped point, using the distance ratio, is a fast and simple solution. The other option would be to apply a vector calculation, which is not a suitable approach. It does not take the spherical shape of earth into account and would not be very accurate. In any case, the distance between two coordinates is calculated using Vincenty's Formulae, which takes the spherical shape into account. Therefore, calculating the distance using the ratio also includes the spherical property. Another option

```
getMappedTrack(racetrackCoordinates, athleteDistance)
  distance = previousDistance = 0
  for (i = 1; (i < path.size() AND distance < athleteDistance); i++)
    previousDistance = distance;
    point1 = gpxCoordinates[i-1]
    point2 = gpxCoordinates[i]
    distance += distance(point1, point2)

  distancePoints = distance - previousDistance
  distanceMappedpoint = athleteDistance - previousDistance
  distanceRatio  = distMpoint / distPoints

  differenceLatitude = point2.latitude - point1.latitude
  differenceLongitude = point2.longitude - point1.longitude
  pointMapped.lat = differenceLatitude * distanceRatio + point1.latitude
  pointMapped.lng = differenceLongitude * distanceRatio + point1.longitude

  mappedTrack = racetrackCoordinates.getElementsFromTo(0, i-2)
  mappedTrack.append(pointM)

  return mappedTrack
```

Listing 4.4: Pseudocode of the Map on Track algorithm.

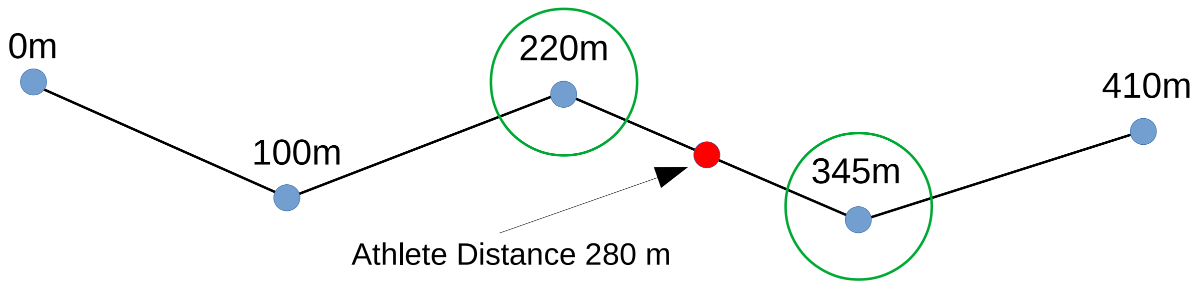


Figure 4.7: The blue circles are the way-points of the racetrack with the distance in meters to the start point above. The current distance of the athlete is 280 meters and the two way-points which are between the distance are marked with the green circles. The red point is the mapped athlete location based on the distance.

would be to calculate the bearing from `point1` to `point2` and adding the distance difference between them in this direction, but this would also take more calculation steps.

4.4.4 Track Mapping Algorithm with Moving Radio-frequency Identification

The implemented mapping algorithm is obtained from Wöllik and Müller [2013]. It is used in a new sports concept, where a car called "moving finish line" overtakes all athletes after a given delay time. This new approach not only includes the position but also the timestamp of athletes to map them. A car passes an athlete and can identify them via the signal of the radio-frequency identification (RFID) device. The system installed on the car stores information about the time and location of an athlete when it passes them. A system where the car collects the information of athletes is also known as a car navigation system. All the data collection is done by a car and because of that, the GPS coordinates are not completely accurate. Therefore, a good mapping algorithm is needed, which decreases the measurement error. The mapping consists of two parts: "indirect mapping", where the RFID tags are mapped to the recorded time and the location values of the car, and the calculation of the estimated cumulative distance, called "direct mapping". The mapping of RFID is not necessary because all tracked athletes use a GPS device, which already includes location and timestamp.

4.4.4.1 Data Preparation

The direct mapping needs two inputs to perform the mapping algorithm:

1. The first part is the predefined racetrack of an event and consists of one or multiple GPX files. Every point of the mapping algorithm consists of latitude, longitude, and the distance from the point to the start. If the distance is not available in the GPX file, it must be calculated. To get the distance of each point on the race track, the implementation uses Vincenty's formulae to calculate the value. Additionally, the algorithm needs a timestamp for every point to project the distance correctly. The timestamp indicates how many seconds it takes to get from the start line to the point with a predefined speed. There are two possibilities to create the timestamps for the GPX file.
   The first option is that the track is recorded properly. This means that all coordinates are recorded using a GPS device and do not deviate from the event track. All GPS coordinates must also have a valid timestamp. Valid means that the points are recorded at racing speed. For example, a racetrack for a running event should not use a car driving at 80 km/h to record the track. A large speed differential between the recorded coordinates and the live coordinates of athletes makes it harder to map the athlete. It may also lead to a false mapping or the algorithm will not provide a result at all.
   Creating the GPX file of the racetrack using software is the second option. In this case, every point stores the latitude and longitude, but the timestamp is missing. Acquiring the timestamp must be done manually. The only information that can be used to calculate the timestamp is the distance. As a reference point, a pace of 5 minutes per kilometer is used. Researching revealed that this pace is a good mean value for competitive running events[30]. To get the timestamp, the algorithm uses the following formula: timestamp = distance $*$ pace $*$ $60/1000$. In both cases the implementation must calculate the distance value manually.

2. The second input is the track of an athlete. Every element of the data set consists of latitude, longitude, and the timestamp. The timestamp of every element is the time difference between the first created coordinate of the athlete and the current element. The algorithm always uses the last created point to map the athlete on the racetrack. To accomplish this, the algorithm calculates the estimated cumulative distance of an athlete.

4.4.4.2 Algorithm

The first step is to determine a window of the racetrack where the current position of the athlete may be. This window is important to map an athlete on the correct part of the track. A problem with using mapping algorithms in general is that they cannot predict the actual position if the track intersects, or if parts of the racetrack are close together. What can happen is that the algorithm maps the current point of an athlete to the wrong part of the track. To avoid this problem, the algorithm uses a so-called time window. First, the algorithm determines the first and last point of a time window. This is done by comparing the timestamps of the racecourse with the timestamp of the athlete. The first point of the window is the point of the racecourse that is smaller than the timestamp of the current position and has the smallest difference in seconds. The second point is exactly the opposite: it is the point of the racetrack where the timestamp is greater than the recorded position and also has the smallest time difference. With the help of those coordinates, it is possible to estimate the distance covered by an athlete. By just calculating the

[30] *Average Half Marathon Pace by Age and Sex.* `http://www.pace-calculator.com/average-half-marathon-pace-by-age-sex.php` (visited on 20/03/2019).

cumulative distance using raw GPS data, it can exceed the total length of a racecourse due to inaccurate GPS records. However, estimating the distance using the time window and the position of the athlete limits the distance between zero and the maximum distance of the track. The projected distance of the current location is calculated using this formula:

$$\begin{aligned}\text{projectedDistance} = {} & (\text{distanceSecondWindowPoint} - \text{distanceFirstWindowPoint})/ \\ & (\text{timestampSecondWindowPoint} - \text{timestampFirstWindowPoint}) * \\ & (\text{timestampCurrentPosition} - \text{timestampFirstWindowPoint}).\end{aligned}$$

If one of two window points does not exist, the distance value is the distance from the existing point to the window point. After calculating the estimated distance, the next step is to determine the pace window. This is done by defining an upper and lower distance bound, which filters the possible points of the pace window. Two scenarios are possible in this case: the first scenario is that the algorithm has not mapped coordinates before and there is no previous knowledge to define the bounds of the pace window. Therefore, the lower and upper bound is the value of the projected distance. If the algorithm has already mapped a coordinate successfully, the upper and lower bound uses this formula:

$$\begin{aligned}\text{lowerBoundDistance} &= \text{projectedDistance} - L * |\text{distanceMappedPoint}| \\ \text{upperBoundDistance} &= \text{projectedDistance} + L * |\text{distanceMappedPoint}|\end{aligned}$$

L is a factor that makes the algorithm more robust. The variable distanceMappedPoint is the distance of a mapped point with the property that the timestamp must be smaller than the timestamp of the athlete. From all these points, the element with the greatest distance is the distanceMappedPoint. The pace window consists of all points of the racecourse, which are between the lower (lowerBoundDistance) and upper bound (upperBoundDistance) distance. Additionally, the window also includes the first point before the lower bound and the first point after the upper bound. Applying this to the first scenario, where the algorithm did not map any previous locations, the pace window includes all points of the race track. Figure 4.8 illustrates an example of a pace window.

Next, the algorithm uses the pace window to find the nearest point to the recorded point of the athlete. To calculate the distance between a point from the pace window and the current coordinate, the algorithm uses the euclidean distance. The nearest point is the point with the smallest euclidean distance to the current athlete position. To project the current point to the track, the algorithm performs an orthogonal projection. The orthogonal projection uses two lines to perform the mapping. One line connects the nearest point with its predecessor of the racetrack and the second line connects the nearest point with its successor. Performing the projection function on both lines results in the relative distance and deviation from the current point to the event track. With the two calculated distance values using the projection function, it is now possible to determine the location of the current point. There are four possibilities (see Figure 4.9):

1. Both projected pedal points lie on their respective line.
2. The pedal point is between the nearest point and its successor.
3. The pedal point is between the nearest point and its predecessor.
4. Both projected pedal points lie outside their respective line.

Based on the four possible locations of the current point, the algorithm calculates the projected distance and the deviation. The deviation value is an indicator for the correctness of the mapping. If the derivation value of a projected point is above a certain threshold, the speed difference between an athlete and the recorded track is too high. This leads to the problem that the pace window is wrong and then the nearest

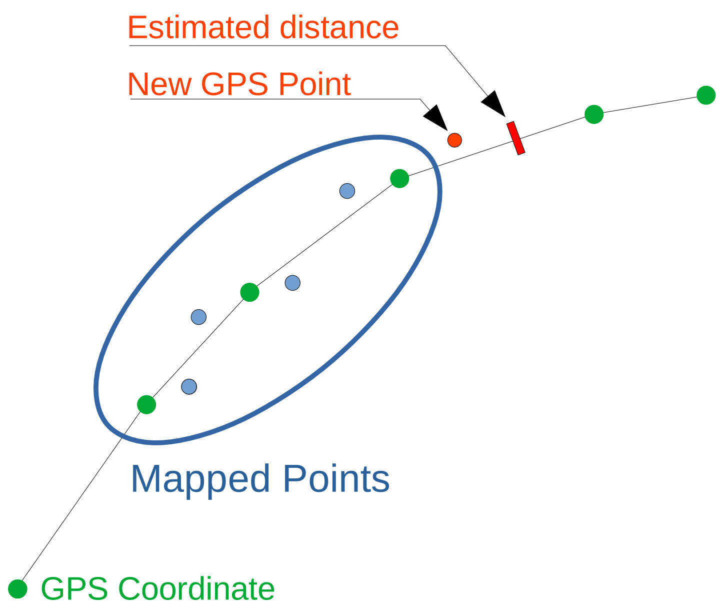


Figure 4.8: An example of the pace window. The green points are the way-points of the racetrack, the blue points are the already mapped GPS points of an athlete. [Image redrawn by the author of this thesis from Wöllik and Müller [2013].]

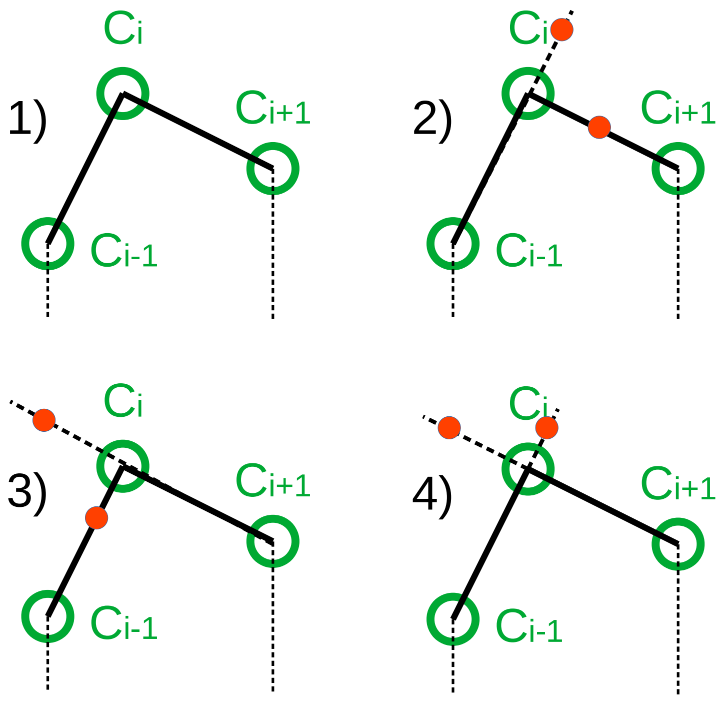


Figure 4.9: The four possible locations of the two projected points (red points). [Image redrawn by the author of this thesis from Wöllik and Müller [2013].]

point to the current location is not in the pace window. To find the correct nearest point, the algorithm must run again where the pace window contains all points of the racecourse. To get the mapped athlete track, the calculated cumulative projected distance is the input for the algorithm described in Section 4.4.3.

4.4.5 No Mapping

The no mapping function does not map any coordinates at all. It is not a mapping function per se, but useful because of a number of factors. First of all, only displaying the raw data can give an overview of the performance of a GPS device. Due to bad signals or a faulty device, the mapping function described above may not provide a valid result. Displaying only the raw data can provide this information. In some cases it is more interesting to use no mapping functions at all. No mapping is also used by Google Map to Road and Google Direction API if the service does not provide a solution. The raw coordinates are also used in sports events such as orienteering races or scavenger hunts. In this case there is no event track available or no roads that can be mapped and mapping is not expedient. It is more interesting to view the athlete track without any mapping.

4.4.6 Comparison and Conclusion

Comparing these five mapping approaches, each of them has its pros and cons. The four mapping algorithms are divided into two categories. The first category includes the Google Map to Road and Google Direction API. These mapping methods do not need the coordinates of the racetrack to perform the mapping algorithm. However, they depend on an external service provided by Google. This service is free for a limited amount of requests; after a certain number, each request is charged. During several tests, it showed that both mapping algorithms have their advantages and disadvantages regarding the mapping. Both of them perform well when the events are in an urban area or a region with paths marked on a map, such as roads, footpaths, or cycle paths. Unmarked paths, such as hiking trails, forest roads or cross-country trails, are not suited because they are not part of the Google Maps road system.
The second category is the mapping algorithm that uses a predefined racetrack, which is stored in the GPX files to map athletes. This includes the Track Mapping Algorithm with Moving Radio-frequency Identification (TMA-MRF) and the Map on Track algorithm. Both of these algorithms map the athlete based on their covered distance. The Map on Track algorithm is much simpler than the TMA-MRF because there is no projection involved. Therefore, it is much faster, however, not as accurate. In contrast to the Map on Track, TMA-MRF provides better results but also needs more computations. The algorithm can avoid wrong mappings but at the expense of computation time. Although it is possible to create the racetrack GPX file using an application and let the server add timestamps by using the average speed, the mapping improves by recording the racecourse in racing speed. Table 4.5 shows the properties of each mapping algorithm.

Property / Algorithm	**GPX File needed**	**Uses external service**	**Keeps information**	**Computation Cost**	**Preferred sports**
Google Map to Road	No	Yes	Partial	Low - data processing for request and response	Cycling, running (on roads that are can be mapped for driving)
Google Direction API	No	Yes	No	Medium - data processing for request, data encoding for response	Cycling, running (on roads that can be mapped for driving and walking)
Map on Track	Yes	No	Mapped coordinate	Low - calculating distance and map one point	All sport events which provide a GPX file
TMA-MRF	Yes	No	Mapped coordinate	High - data preparation, calculate pace window, orthogonal projection and estimate distance	All sport events which provide a GPX file
No Mapping	No	No	Yes	None	All of sports

Table 4.5: Comparison of the implemented mapping algorithm.

4.5 Charts

Another requirement is to visualize the performance of athletes over the course of an event, which is best done using charts for this specification. A chart is either created for a specific athlete or for the racetrack. The goal of the chart implementation is to create a template that is flexible and easy to extend. Once again it should be easy for other developers to implement new charts and use any chart type. The server is responsible for processing the data for the client, which uses the data to render the chart on the web site.
A chart consists of multiple components:

1. The type of chart, for example, Pie Chart, Bar Chart, Line Chart.
2. A data table, which contains all the data for the visualization.
3. Formatting and styling options for the chart.

The client uses the Google Chart library to render charts, so all chart types are provided by the library[31]. The implementation makes it possible to use any chart by simply capitalizing the first letter of each word and putting them together, such as using a bar chart by using the string `BarChart`, or visualizing them as a stepped area chart by using the string `SteppedAreaChart`.

[31]Google, *Chart Gallery*. `https://developers.google.com/chart/interactive/docs/gallery` (visited on 20/03/2019).

To create a diagram, the data must use a specific container. The software structure used for the data storage is similar to a table on a spreadsheet. A column specifies the content of the rows and a row is described by three properties. The ID of the column, which must be a unique string, helps to access a column without knowing the exact order or position of the column in the table. Besides the ID, a column also can set a string that describes the values. The description is displayed if the user moves the mouse over the chart, for example, by moving the mouse on a bar of a bar chart. Each column must define the value type and use the supported types, otherwise the library does not know how to handle or display the data. The following data types are compatible with the Google Charts Library:

- Boolean: The value of the cell can either be the value true or false of the type Boolean.
- Number: The value must be a number of the type Byte, Integer, Short, Long, Double or Float.
- Text: Any type of string is acceptable.
- Date: The date needs an instance of the class GregorianCalendar. For example, the date 27.12.2008 18:13:41 is created new GregorianCalendar(2008, 12, 27, 18, 13, 41). The date time however only uses the day, month, and year.
- DateTimes: Needs also a GregorianCalander class instance for the values of the column. Taking the example used for the Date, the DateTime also includes the hours, minutes, and seconds for the representation of the data.
- TimeOfDay: As well as the Date and the DateTime, the TimeOfDay also needs an instance of the class GregorianCalander. In contrast to the other two date formats, the TimeOfDay only uses the hour, seconds, and minutes. For the chart the day, month, and year is not relevant.

Each table consists of a minimum of two columns. Depending on the used chart type, the structuring of the table can change. For example, a line chart needs at least two columns. The first column represents either the group labels of the data or the values along the x-axis. If the first column contains a group label, the data type is a string, otherwise it is a number. The second and following columns represent the data of the different instances. These columns contain the values on the y-axis.
Each row consists of an array of cells. The number of columns defines the number of cells in a row. The value of each cell must also match the defined type of the column description. There is no limitation for the amount of rows in the data set.

The third property of each chart is to change the formatting and configuration options. By default, each chart includes the possibilities to add a title to the chart. The position of the title can be played in two different regions: placing the title inside the chart area using the value `in` or outside the chart area setting the value to `out`. The third option is to not display the title of the diagram, using the option `none`. The second modifiable part is the legend, of which multiple components are configurable. The legend can be placed in four different regions:

- `Left`: The legend displays on the left side. If a chart has a description on left side on the x-axis, it is important, to move it to the right side.
- `In`: The legend position inside the chart, in the top left corner.
- `Right`: Legend shown right outside the chart area. Again, it is not compatible, if the x-axis description shown on the right side.
- `Top`: Placing the legend on the top of the chart.
- `Bottom`: The legend is below the chart.
- `None`: Does not display the legend.

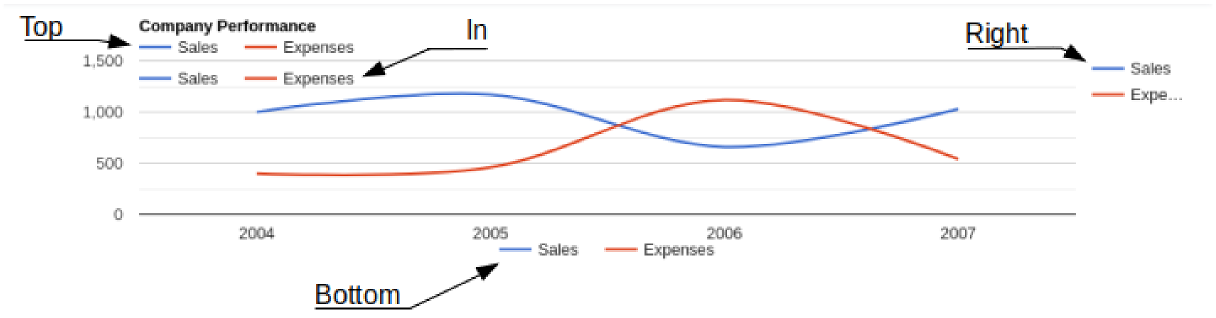


Figure 4.10: The legend positions of the chart. `left` is not possible for this chart.

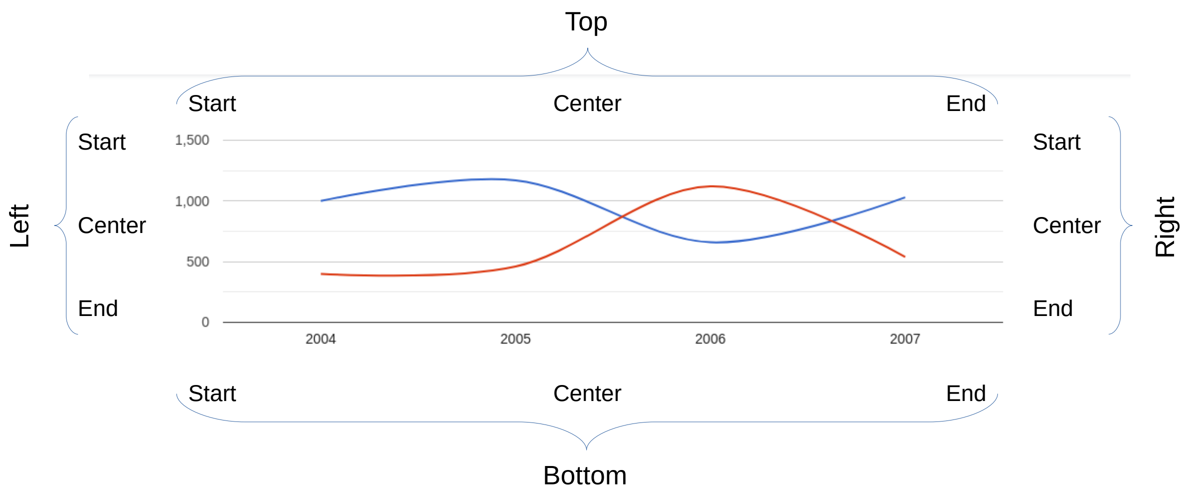


Figure 4.11: The legend alignment for all four positions.

The default location of the legend is on the right side, if none of these options is selected. Figure 4.10 shows all positing regions for the legend in a chart.

It is also possible to automatically align the legend based on three alignments. The options are `start`, `center` and `end`. Each chart has a reserved area for the legend. Start, center, and end indicates the position in this area. Depending on the chart, the default positing area of the legend is either vertical or horizontal and based on the four alignments: top, bottom, left, or right. If the chart positions the legend on the right or left side, start places the legend on the top and end on the bottom. If the legend is positioned left or right, the start and end are at top or bottom of the area. The default value depends on the chart. For a chart with a bottom legend, the default alignment is center and for other ones the default value is start. Figure 4.11 shows the areas of the different alignments.

The options explained above are properties that all charts include. More specific chart types offer more features to modify. The following features are included in the line chart. It is possible to make the lines displayed on the chart smoother by setting `curveType = function`. The smoothing function tries to filter out noise data and creates an approximated function.

Another modification that can be applied to charts is the axes' customization. Axes formatting is supported by several chart types such as bar charts, line charts, column charts, area charts, scatter charts, combo charts, candlestick charts and stepped area charts. These charts use the axes to display the dimensions of the data. The major axis can either be discrete or continuous. An example of discrete data displayed in a bar chart is the population of the states in Austria. Discrete data has a finite number of values, also called categories. Continuous data has an infinite number of possible values. The first column of discrete data is a string and for continuous data it is either a number or a date format. Both axes, vertical and horizontal, can format the numbers of each axis. These formats are supported to customize the axes:

- `None`: Displaying the raw data without using formatting, for example, 7500000.
- `Decimal`: Numbers displayed with thousands separators, for example, 7.500.000.
- Scientific: Displays numbers in scientific notation, for example, 7.5e6.
- `Currency`: Displays numbers in a currency format, for example, € 7.500.000.
- `Percent`: Numbers displayed as percentages with thousands separators, for example, 7.500.000.000%.
- `Short`: Numbers displayed abbreviated, for example, 7.5 M. 500000 is displayed as 500K.
- `Long`: Number display as full words, for example, 7.5 millions. 5000000 is displayed as 500 thousand.
- `Date`: Dates and time, can formatted using the ICU date and time format pattern[32].

Another feature is to use scale function on both axes. There are two possible logarithmic scaling functions. The first one is a standard logarithmic scaling, which filters out negative and zero values. The second logarithmic function it called `mirrorLog` and it includes negative and zero values into the plot. The logarithm is not defined for zero and negative values. Therefore, the logarithm for negative numbers is defined as $-\log(|x|)$. Numbers that are close to zero do not use any scaling function.

An additional customization is to change the values on the x and y axis. The option offers the possibility to replace the automatically generated ticks with custom ones. Valid data types for the ticks are `Numbers`, `Dates`, `DateTimes` and `TimeOfDay`. The manually created ticks are an array that consists of multiple values. Using an empty list however, removes all ticks of the modified axis. Customizing ticks is only possible for continuous data sets.

Each axis can set a title which is placed on the left and bottom of the chart area. In the implementation, it is possible to use the features of i18next. The title is part of the data sent to the client, which is rendered as an SVG element in the browsers. Using i18next does not support parsing of SVG elements. For that reason, the server takes responsibility to translate all text elements used in the chart. Every chart has its own name space for the translation. The used notation for the prefix is `backend.chart.nameofthechart`. For example, a new chart that calculates the burned calories of an athlete uses the prefix `backend.chart.calories`. The name of the title uses the prefix `backend.chart.calories.title`. Accessing the title of both axes requires the prefixes `backend.chart.calories.vaxis` and `backend.chart.calories.haxis`. The description of the major and minor axes uses for example, `backend.chart.calories.columnCalories` and `backend.chart.calories.columnTime`.

The following charts are implemented in the current state of the project:

- Speed Chart
- Pace Chart
- Altitude Profile Chart

All of these implementations must return an object of a chart using a function called `getChart`.

[32]International Components for Unicode, *Formatting Dates and Times*. `http://userguide.icu-project.org/formatparse/datetime` (visited on 20/03/2019).

Furthermore, each implementation must define the prefix it uses for the translation text of titles and column description, as mentioned above. The implementation can modify the customization options to change the styling of the chart.

4.5.1 Speed Chart

A speed chart is a line chart which visualizes the speed of an athlete during the course of an event. To calculate the data, the implementation from Section 4.3.3 is used. The chart uses the GPS data of an athlete, sorted by the creation date as input. The data table consists of two columns: the first column stores a timestamp and the second column stores the calculated speed. To create the data table, the algorithm iterates over the GPS coordinates and in every iterative step the timestamp and speed is calculated. The speed is the time difference in seconds between the first GPS coordinate in the list and the current iteration. The second cell of the row stores the speed between current and previous point of the iteration. To automatically format the numbers as a time format, the y-axis uses the `TimeOfDay` format. The y-axis that represents the speed in km/h is displayed as a decimal format. Figure 6.30 shows a speed chart.

4.5.2 Pace Chart

The pace chart is similar to the speed chart. It uses the GPS data of an athlete to generate the data table. The calculation of the pace uses the method discussed in Section 4.3.5. The data table consists of two columns: the first column stores the timestamp of the calculated value and the second column holds the pace of an athlete. Once again, calculating the pace iterates over all GPS coordinates, using the current element of the iteration and the previous to calculate the values. The timestamp of the pace is the time difference between the first GPS record in the list and the current iteration. Both axes use time units for the presentation of the data. The pace has the unit minutes per kilometers and the timestamp uses seconds. Therefore, both axes use a `TimeOfDay` as a format. A speed chart is shown in Figure 6.31.

4.5.3 Altitude Chart

The altitude chart displays the altitude profile of GPS records of athletes, or the provided GPX file of an event can do so. The data table of the chart consist of two columns: the first column stores the distance to the start point and the second column stores the elevation on the given distance. An iterative algorithm loops over all GPS data of the record. In each iteration step, the distance between the previous and current element is calculated using Vincenty's formulae, as discussed in Section 4.3.1. The calculated distance is added to the total distance in every iteration step. The summed up distance of every iterations step is the value of the first column. The second column stores the altitude value extracted from the GPS coordinates. The unit of both columns are meters, therefore both axes are formatted as numbers. Figure 6.13 shows an altitude chart.

4.6 Deployment

The deployment process is responsible for making the software executable on the user's system. Executing the application should not require any complicated steps or configurations. For users it is important that the application runs out of the box. To make this possible, the executable Java application is a so-called "fat jar". A Java Archive[33] (jar) includes the classes, resources, and metadata of the software which is needed to execute the application. By default, jar files do not include the required dependencies such

[33]Oracle, *JAR File Specification.* `https://docs.oracle.com/javase/6/docs/technotes/guides/jar/jar.html` (visited on 20/03/2019).

key	data type	descriptions
server.port	integer	The external port to access the server
jdbc.driverClassName	string	The vendor of the external database
jdbc.url	string	URL to the database
jdbc.username	string	Username for database login
jdbc.password	string	Password for database login
server.debug	Boolean	If true, dummy data is shown on the server for testing
server.static.minimize	Boolean	A folder which contains the minimized JavaScript files can be used

Table 4.6: The entries of the server property file which can be changed by the user.

as used libraries. Coping the file and executing it on another computer would not be possible because of the missing dependencies. A fat jar on the other hand, includes all these dependencies and works like a portable app. However, the disadvantage is that the file size is bigger because of the included dependencies.

The execution must also initialize the environment, if it is executed for the first time. These initialization steps include copying the files for the web application, creating folder structures to store GPX files or event information. It also copies Java property files[34]. The purpose of a such a file is to configure the application without changing the code. A Java property file consists of key-value pairs, which the user can change. The values that are changeable are shown in Table 4.6. After entering the database information the application can be executed without further configurations. Besides the server configuration file, another one is also created for the events. This file stores the distances of all GPX files of the event track, to avoid redundant calculations. Furthermore, if there is no GPX file available for an event, a static distance can be entered that the statistics can use for the calculation.

Minimizing and compressing JavaScript files is a common technique and used by nearly every web site. The web page consists of approximately 1.4 Megabyte uncompressed JavaScript files. Tools such as UglifyJS[35], Minify[36], Closure Compiler[37] or JSCompress[38] can optimize JavaScript files by simplifying the code and removing unnecessary lines and compress files, by removing spacing and shortening variable or method names. Using these optimizing tools decreases the file size up to 70%.

[34]Oracle, *Properties (Java SE 10 & JDK 10).* `https://docs.oracle.com/javase/10/docs/api/java/util/Properties.html` (visited on 20/03/2019).

[35]Bazon, *UglifyJS - JavaScript parser, compressor, minifier written in JS.* `http://lisperator.net/uglifyjs/` (visited on 20/03/2019).

[36]Mullie, *Minify - JavaScript and CSS minifier.* `https://www.minifier.org/` (visited on 20/03/2019).

[37]Google, *Closure Compiler.* `https://developers.google.com/closure/compiler/` (visited on 12/04/2019).

[38]JSCompress, *The JavaScript Compression Tool.* `https://jscompress.com/` (visited on 12/04/2019).

Chapter 5

Representational State Transfer

The representational state transfer (REST) interface is a core component of the application. It provides an interface, which is accessible for any devices via the internet. The client uses the interface to receive or transmit the data and is also needed to create the view. Nearly any device can easily access the REST interface, which makes it easy to share the produced data. To access the REST interface, no heavyweight library or extension is needed, because is uses the Hypertext Transfer Protocol (HTTP). REST is accessed using a Uniform Resource Identifier (URI) and it is possible to restrict the path to certain user groups using Java Servlets (see Section 4.1.3).

Using the interface should be possible on any operation system. Another requirement is that it should possible to upload data to the server. It should be possible to restrict the access for specific user groups, for example, guests, registered users and admins. The received data should be easy to process without converting it. Considering these specifications, the design decisions were narrowed down to two possible solutions. On one hand, there is the possibility to implement interfaces using REST, on the other, WebSockets would also be an efficient solution.

5.1 REST vs WebSocket

One of the major design decisions was to determine which technical approach to use for accessing data or transmitting it to the server. There are two popular solutions for this problem, which are used by many other applications. The papers of Pautasso et al. [2008] and Pautasso and Wilde [2010] helped to make the decision.

WebSocket [Fette and Melnikov 2017] is a network transmission protocol based on the Transmission Control Protocol (TCP). WebSocket is a newer protocol, standardized by the Internet Engineering Task Force (IETF) in 2011. Thus WebSockets use the HTTP ports 80 and 443, the protocol is not an HTTP. Other than the HTTP, WebSockets use a single TCP connection to communicate. The client and server can communicate in both directions via the single TCP connection. Then the client can send or receive data to the server and vice versa. The communication between server and client in both directions is called full duplex. To establish a connection, the HTTP is used and upgraded to a WebSocket. One of the advantages of a regular HTTP connection is that, to request data from the WebSocket, only one connection between the server and client is established. After the initial handshake between the server and client, every request and response will be sent using this channel. Establishing only one connection with one handshake results in a lower data traffic.

Therefore less data is used during the lifetime of the connection. The connection between the client and server remains open after the handshake. This allows the server to automatically send updates to the client. Therefore, the server has to implement a notification system, which sends updates to the clients. Additionally, the server must implement a logic which checks if the data in the database has changed. In addition to these two task, the server must keep track of the clients' session. The server must analyze which clients are still subscribed to a service and which connections no longer exist. A major disadvantage of using WebSockets is that linking to resources is not possible. WebSockets uses a URI to establish the connection between server and client, but it is not possible to create a link to a certain resource. Accessing resources is only possible via the TCP channel and not shareable via a link, unlike a REST interface.

The second solution is to use the REST programming paradigm. The REST services emulate the behavior of the internet. Every web server which can host websites via HTTP, such as that of the TU Graz (`www.tugraz.at`), is already compatible to provide a REST web service. Any device that is able to make an HTTP request and parse the server response can use REST interface services. REST also uses the TCP to establish a connection to the server. In contrast to the WebSocket, the connection between the server and the client closes after the client receives a response from the server. It is important to mention that the server does not store any information about the session or other client information. Every request creates a new connection between the client and the server and, unlike the WebSocket, each resource has its own URI. This makes it shareable and accessible for all users via the internet. It is much easier for clients to request or send data to a server via a REST API rather than using WebSockets. To request data, the client only needs the URI and processes them, whereas with a WebSocket, the client must establish the connection and then request the resource. REST makes the access to the resource for other applications much easier than using WebSockets. It makes it less difficult for other applications or clients to access the data and process them, which is important for further usage of this project. REST is well tested over time and due to this, this solution is well documented. The aforementioned reasons have led to the conclusion that REST interfaces are the better choice.

5.2 Resources

In the context of REST, the term resource is very important. A resource is something a user wants to share via a link so that the received data from the resource can be edited. The user should be able to receive the data and make further use of it. A resource in the computer world is mostly data accessed from a database, data produced by an algorithm or data read from a file. This data is accessible via a uniform interface. The uniform interface uses the HTTP and is accessed via a URI. Instead of calling function names like getList() or setName(), resources using the HTTP methods as described in Section 5.5.7. The resource representation is up to the developer and can choose the identifier the way he likes. Resources should be accessible in a way that the user can change the representation or filter the data, without changing the resource itself.

5.3 Resource Types

Basic architecture types developed over time show how to model an interface for REST. Nowadays these types are well known and an agreed standard in regard to design resources. Usually, the following four resource types are used:

1. Document
2. Collection
3. Store
4. Controller

The Document is the basic concept of a resource. It maps to a single object, such as a single object from the data collection or a record of the database. The document type is the base for the other resource types, which are in a way variations of it. The URI to a document resource is not necessarily the tail of the path. It is possible that a URI to a document can have multiple sub-categories, which we will refer to here as children resources or elements.

Some examples of document resources:

- `http://api.mywebsite.com/events/graz` (returns information about the city of Graz).
- `http://api.mywebsite.com/events/graz/runs/grazathlon` (returns information about the event Grazathlon).
- `http://api.mywebsite.com/events/graz/runs/grazathlon/athletes/runner1` (returns information about runner1 of the Grazathlon).

The Collection resource can be compared to a list of objects in a computer program. It returns all instances from a certain object collection. Every instance can be a parent element and contain a list of children elements. The instances can contain information about how to access a child element, for example, via an identifier (ID). Collections are not necessarily static. Depending on the design choices, the user can add, remove, or change collections. These are some examples for URIs based on the scenario above:

- `http://api.mywebsite.com/events/` (returns a list of events).
- `http://api.mywebsite.com/events/graz/runs/` (returns a list of all running events in Graz).
- `http://api.mywebsite.com/events/graz/runs/grazathlon/athletes/` (returns a list of all runners attending in the Grazathlon).

The store resource is like a temporary collection, which a user can change. They can create new resources or delete elements from the store, for example, and it does not affect any other resources or create new URIs. It is comparable to the home screen of a smartphone; a user can reorder the apps, put them into a folder, or remove them from the home screen without actually deleting them from the phone.

A controller resource model is an explicit function call executed via a REST API call. The client sends a request to re-perform a procedure on the server, for example, resend notifications to all users. The controller type is often the last segment of the URI path sequence. Therefore, no child resources follow.

5.4 URI

Uniform Resource Identifier (URI) is one of the most important parts when using the web. URIs are very simple, but still one of the main reasons the internet works as we know it. URIs developed over time and were formerly known as Web addresses, Universal Document Identifiers and Universal Resource Identifiers [Berners-Lee 1994]. Finally, in 1995, the Internet Engineering Task Force released the URI standard [Berners-Lee et al. 1998], and it is a combination of the Uniform Resource Locators (URL) [Berners-Lee et al. 1998] and Names (URN) [Sollins and R. Fielding 1994]. The URI standard was finalized in 2005 and is defined in the RFC 3986 [Berners-Lee et al. 2005].

The syntax of a URI consists of five parts as shown in Figure 5.1:

1. The **scheme** must be set and starts with a letter followed by any combination of letters, digits, period, hyphen, or plus sign. A colon follows the scheme. Well-known schemes are, for example, http, https, ftp, file, or ssh.
2. The **authority** component is an optional part of the syntax. The authority element is placed after a double slash symbol "//". The user info part is also optional. It consists of a username followed by the password. A colon separates the username from the password. The @-symbol closes the sub-component of the user info. An optional host element follows after the user info. The host

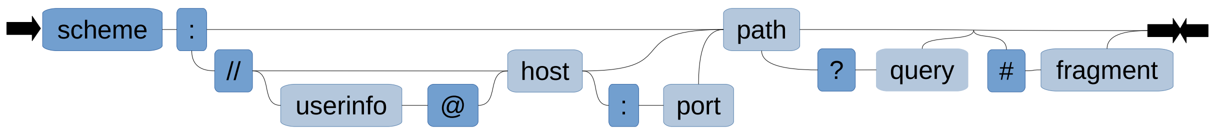


Figure 5.1: The Syntax diagram of the URI. The dark blue colored labels are predefined parts of the URI and the light colored ones are the variables.

sub-component is either a registered name or an IP address. The last sub-component is the port number that goes ahead of a colon. URI schemes like FTP or HTTP must have an authority element.

3. The **path** helps to define the resource. It can consist of multiple path components separated by a slash (/). A path or a segment in a path can be empty which results in two consecutive slashes (//). If the authority element is set, the path component may be empty, or it can begin with a slash. If the authority component is not present, the path cannot begin with an empty segment (//). In this case, the syntax would be mistaken with an authority element. The path is limited by the query, the fragment, or the end of the URI.

4. **Query**, also known as a query string, is an optional component of the scheme. It starts with a question mark symbol, followed by a sequence of key value pairs. The delimiter of a key value pair is usually an ampersand or a semicolon. For example, key1=value1&key2=value2 or key1=value1&key2=value2. The server uses query strings to filter the produced data of a resource.

5. The **fragment** component is also an optional component preceded by a pound sign. It is used to refer to another resource within the product resource. The most common usage of the fragment component is in an HTML document, to scroll to a certain position in the document.

Here are some examples for URIs:

1. `ssh://user:password@pluto.tugraz.at:2222/path/to/myfiles?x11=display-location`

Schema:	ssh
Userinfo:	user:password
Host:	`pluto.tugraz.at`
Port:	2222
Path:	`/path/to/myfiles`
Query:	`?x11=display-location`
Fragment:	-

2. `mailto:max.mustermann@tugraz.at`

Schema:	mailto
Userinfo:	-
Host:	-
Port:	-
Path:	`max.mustermann@tugraz.at`
Query:	-
Fragment:	-

3. . `https://google.com/search?q=tugraz&lr=lang_de`

Schema:	https
Userinfo:	-
Host:	`google.com`
Port:	-
Path:	`/search`
Query:	`?q=tugraz&lr=lang_de`
Fragment:	-

4. `https://en.wikipedia.org/wiki/Graz_University_of_Technology#History`

Schema: :	https
Userinfo:	-
Host:	`en.wikipedia.org`
Port:	-
Path:	`/wiki/Graz_University_of_Technology`
Query:	-
Fragment:	#History

5.5 Restful

Creating web services using the REST programming paradigm involved following guidelines. These guidelines consist of six points and a web service that fulfills all these points named RESTful [R. T. Fielding 2000; Riva and Laitkorpi 2009].

5.5.1 Client-Server Architecture

The client server architecture is a network architecture style. It consists of a server which offers services to the clients and listens to their requests. A client triggers a service from a server by sending a request, which the server can then perform or reject. The client-server architecture of REST leads to a separation of concerns. To handle the client's requests is the server's only responsibility. The client is in charge of processing the data and displaying it for the user. For example, the client site is responsible for creating a dynamic view with the data from the server. The task of the server is to keep the REST interface accessible. It does not matter for the server if the client changes their application.

5.5.2 Addressability

The addressability of a web service via a URI is comparable to the file system used in a computer. For example, the path to a file is easily addressable by multiple applications. The user can directly access the file by just copying the path and using it within the application instead of navigating through the whole file system again. The same concept is used to access resources over the internet, of which addressability is one of the major features. It ensures that the client can access a URI and is able to access the same web service in the future again. Another advantage is that the user can share or bookmark the web service via the URI and address it later. Sharing data via the internet is made easier by using web resources which are addressable for everyone rather than downloading it locally and sending your copy to another person.

5.5.3 Statelessness

Stateless means that every request send by a client contains all the information needed for the server to create a response. The server treats every HTTP request independently. The client does not have to force the server to change its state or for that matter its data to produce a certain outcome. Every state the server can produce should be available as a resource via a URI. If the server wants to keep track of a client, the client has to store information about the session on their device. However, in most instances, cookies, or the local browser storage is used. Every request the client makes has to include their session identifier. This identifier is used by the server to access data which is connected to client. The decision not include the state in the protocol makes it less prone to errors. Furthermore, an interaction between a client and the server does not last longer than one request. The server does not have to worry about timeouts or loss of information because all the information needed is stored by the client and adds them to the request. Another key aspect of statelessness is that it is far easier to increase the number of servers used by an application. Since every request is independent, the servers do not need to interact among each other. An operator of a web service can add a few new servers to increase the performance and does not have to worry about any communication problems between the servers.

5.5.4 Cacheability

Using HTTP makes it possible to use caching. The client can temporary store data from the server and the response must label whether it can be cached or not. If it can be cached, the client can reuse the cached data later for identical requests. Data caching improves the efficiency and reduces the amount of data transferred between client and server. The disadvantage of using caching is that the cached data of the previous request can differ significantly from the current state on the server. The receiver must decide which resources to cache, and which not.

5.5.5 Layered System

The layered system architecture, also known as multi-tier architecture, is a client-server architecture with components separated into layers that each have a specific role. These roles are, to name a few, presentation layer, business logic and database tier. A layer receives data from the previous layer, processes the data, and sends it to the next layer. Each layer provides interfaces for the neighboring ones. A layer can change its implementation without affecting other layers.

5.5.6 Representations

Data transfer that is produced by a resource can be represented in multiple ways. The resource defines the representation types available to the clients, which depend on the service. For example, a resource which produces a graphical chart is available as a scalable vector graphics (SVG) or portable network graphics (PNG). The client and the server must negotiate which representation type should be used, and then the client sends a request to the server. This request can contain the following properties as defined by the RFC 7231 [Phillips and Davis 2009]

- Accept: The accepted media type. It defines which format the server should transmit. For example, JavaScript object notation (JSON), extensible markup language (XML), portable document format (PDF), hypertext markup language (HTML) and so on.
- Accepted-Charset: Specifies the encoding scheme of the textual representation of the data. Possible character encodings are UTF-8, UTF-16 or ISO 8859.
- Accept-Encoding: Gives information on whether the server should compress the response data. The client sends a list of supported compression methods.
- Accept-Language: The client can send a request to use a specific language. The language settings are usually extracted from the web browser.

To perform a negation successfully, the client and the server must agree to the listed properties above. If the negotiation is not successful, the client cannot access the resource. As stated in Sections 5.5.2 and 5.5.3 to keep the service RESTful, not all information should be hidden in the request header. The language can be part of the URI to make web services more addressable. For example, the language of `google.at` can change via the URI. For the `https://www.google.at/?hl=en` produces the English version and `https://www.google.at/?hl=de` a German version.

5.5.7 Uniform interface

The uniform interface is one of the most important constraints for RESTful services. Using the HTTP, there are limited methods that can be applied to the resources. The four most used are HTTP GET, HTTP PUT, HTTP POST and HTTP DELETE. HTTP GET retrieves a resource from the server. To create a new resource, both HTTP PUT and HTTP POST are used. To modify a recourse, HTTP PUT method is applied to an existing URI. Deleting an existing resource uses HTTP DELETE.

A major advantage of relying on HTTP methods is the achievement of two properties: safety and impotence. Using the GET and HEAD method correctly to receive data from the server will not change the server's state. This means when the client performs a GET or HEAD request, the result will stay the same. Executing the same request multiple times has no impact on the outcome of the response. There are no changes on the server such as creating or modifying a resource or creating a new directory. Impotence in the computer world means that if the computer executes a program multiple times, the result will always be the same. Impotence is important for the RESTful service because these services are used over unreliable networks. If a client sends a GET request to the server, it is possible that they do not receive a response from it. Therefore, it is important for the interface not to change. The client sends multiple GET requests to a URI and still can expect the services not to change. It is important to implement REST APIs in such a way that HTTP methods interpret it correctly. For example, a GET request should not change any resources on the server.

5.6 Rules for REST API Designing

REST APIs use URIs to access resources. Using a standardized name convention is necessary to provide consistency throughout all URIs. Masse [2011] and Sohan et al. [2017] discuss multiple rules to create a uniform URI representation.

The usage of hyphens makes a long URI more readable for the human eye. A sequence of words that would usually be separated with spaces may use hyphens instead. They avoid using underscores as separators though, seeing as editors and browsers often underline URIs to indicate the text is a clickable link. This can confuse the user because the underline can hide underscores.

According to RFC 3986 [Berners-Lee et al. 2005] , lower case URI paths are recommended. Except for the scheme and the host, the URI is case sensitive. For example, the URI `http://my.restapi.com/My-Resource/resource` differs from the URI `http://my.restapi.com/my-resource/resource` and produces a different outcome and could lead to confusion.

The return format of a resource should not be included in the URI. For example, if the user wants a JSON format as a return type and adds the return type as a fake file extension to the URI `http://api.mywebsite.com/events/graz.json`. Using the functionality of HTTP instead is a better solution. To get the desired return type, one must utilize the Content-Type header and set the desired type form for the Media Types.

Using a consistent prefix to access the REST API is achieved by utilizing a constant sub-domain or the path is preceded by a constant string. As an illustration, the web page `http://mywebsite.com` can use `http://api.mywebsite.com/` or `http://mywebsite.com/api/` as the base URI. This constant URI is called root resource or docroot of the REST API.

There are rules that are applied on the types described in Section 5.3. Document resources should use single noun or noun phrase names in the URI path. Collections should use a plural noun name. In addition, the noun should give a clear description, about what objects a collection contains. The same rule applies to the store resource type. Controller names should use a verb. The verb should clearly express the action performed on the server.

URI path segments can be static or variable. Static paths have fixed names, which the designer chooses. Variable path segments use numerical or alphabetic identifiers to create a unique URI. The URI has a fixed template with variables. In most cases, the client substitutes the variables to create a URI. An example of a URI template with variables could be as follows:

- `http://api.mywebsite.com/events/{locationId}/runs/{eventId}/athletes/{athleteId}`

An illustration of the unique URI after replacing the variables is shown below:

- `http://api.mywebsite.com/events/graz/runs/grazathlon/athletes/runner1`
- `http://api.mywebsite.com/events/2/runs/1023/athletes/782`

In the first example, the `locationId` is replaced with the string `graz`, the `eventId` with `grazathlon` and the `athleteId` with `runner1`. This example uses an alphabetic identifier to find unique instances in a collection. In contrast to example one, example two uses numerical identifiers to identify instances. This is a more common approach because in most cases, the data is stored in a database, which uses auto-incremented integers to make records unique.

Elements of a collection or store are filtered by using the query fragment of the URI syntax. The API must define the search criteria which can be used to filter the collection. Every search criteria is represented by a unique string defined by the API and the client must set the values in the query fragment. Values must use the correct data type to successfully receive a server response. Valid types for the values can be a number, string, or more complex types such as a date or a list. The complexity of the filtering options are up to the designer.

Another usage of the query fragment is to sort instances of a collection. Common options are sorting by alphabetical order, sorting elements by creation date or instances by an attribute. Sorting a collection by an attribute is similar if not identical to products that are sorted by their price or countries sorted by population. The sorting options often include the possibility to sort the collections in ascending or descending order.

Queries are not just used to filter or sort collections and stores, but also to paginate results. To paginate a result, two parameters are used: the `pageSize` indicates the maximum number of elements of the result and the `pageStartIndex` defines the index of the first element. Almost every website that lists a large number of elements uses pagination, such as Amazon, IMDb, or Google Search. For example, this URIs uses the query fragment for filtering and paginating:

1. `https://www.google.at/search?num=20&start=40&q=tu+graz`
2. `https://www.imdb.com/search/title?genres=action&sort=rating,desc&start=20`

In the first example, the key `q` is used to filter after the string "tu graz". In this case, the result includes all instances that contain the string. The result counts twenty elements restricted by the page size key `num` (`?num=20`). The starting index of the first element is at 40, set with the key-value pair `&start=40`. This configuration produces the same outcome as when a user navigates to the third page of search results.
The second example includes all three properties as explained above. The result contains all movies included in the action genre (`genres=action`). Additionally, the movies are sorted by their rating in descending order, which means the highest rated movie is the first element. The page size is a constant with the value 50 and therefore not included in the query string. The key `start` represents the starting index of the first element, which is set to 20.
These two examples are not exactly using REST interfaces but still accessing a resource via a URI. The resources produce an HTML view instead of a well readable format like JSON, but it gives a good example of the usage of the query fragment.

5.7 Resource Specification

The implemented resources on the server are accessed by the client, as discussed in Section 6.1. Figure 5.2 shows a diagram of all resources and the corresponding request types. The implemented resources have to map the model of statistics, charts, athlete locations and events from the database access object to the output format. This is done using the so called data transfer object (DTO). The mapping is done using GSON (see Section 4.1.7) which produces the JSON output.

5.7.1 Event Interface

Description: The interface to access the event data, filter them and show specific events.

URI: `REST_URL/events`

Request Types:

- **GET**: **Description**: Returns a list of all events, sorted by event start time in ascending order.
  Produces: application/json
  HTTP status code: 200: OK - The request has been responded as expected.

Output: The produced JSON object of a event holds the following attributes with the datatype in parenthesis:

- `id`: ID of the event (number).
- `name`: Name of the event (string).
- `startTime`: Timestamp of start time (number).
- `endTime`: Timestamp of end time (number).
- `isEnabled`: True if the event is public or false if the event is private (Boolean).
- `description`: A description of the event (string).

5.7.1.1 Event Query

URI: `REST_URL/events/query`

Request Types:

- **GET**: **Description**: Returns a list of events which match the query parameters, sorted by event start time in descending order.
  Query Parameters:
  `from`: Lower bound for event date (number)
  `to`: Upper bound for event date (number)
  `limit`: Size limit of the event list (number)
  `offset`: Starting index of the first element (number)
  Produces: application/json
  HTTP status code: 200: OK - The request has been responded as expected.

5.7.1.2 Event Object

URI: `REST_URL/events/{eventId}`

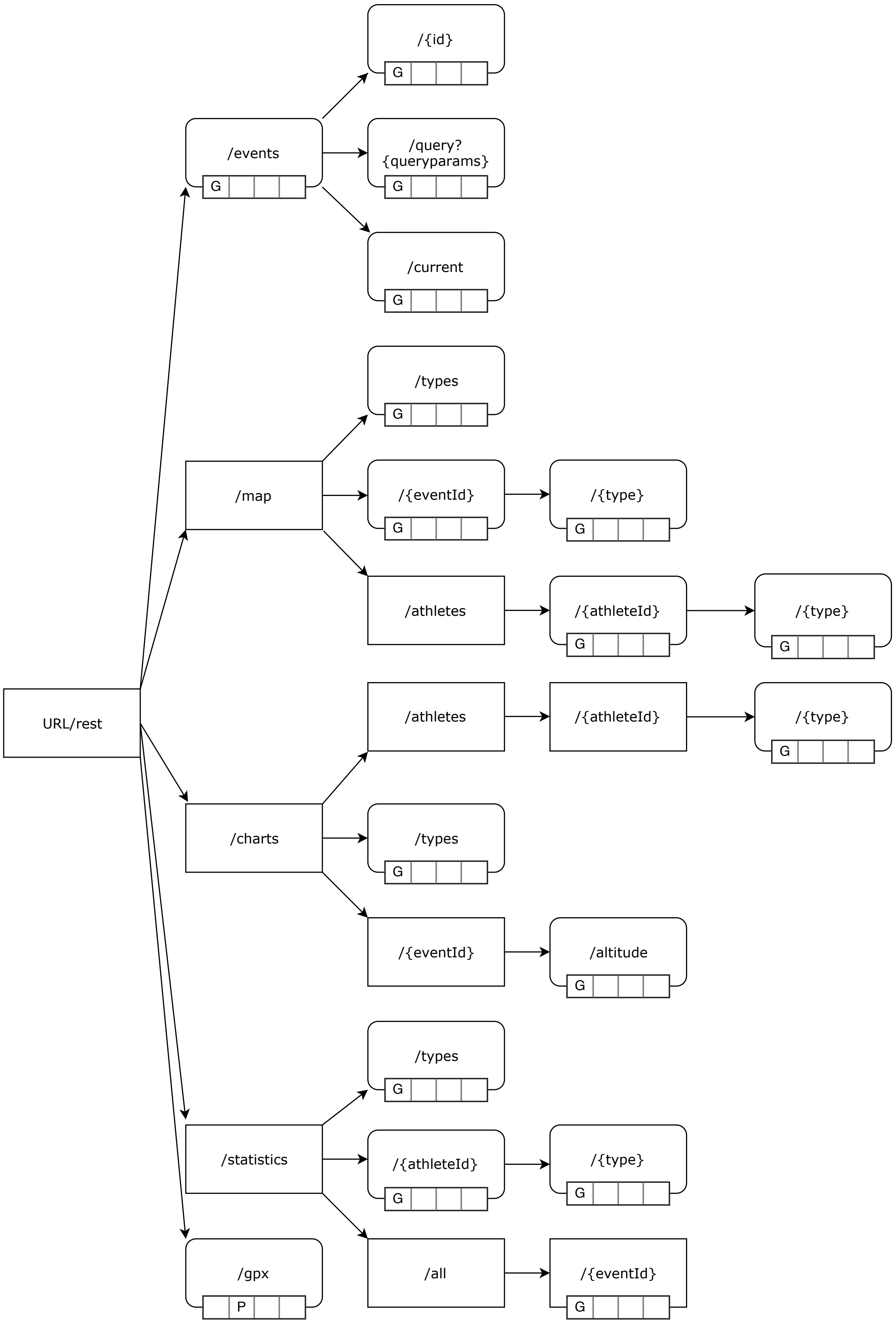


Figure 5.2: A tree diagram showing the path to every resource specified in Section 5.7. Path elements with rounded corners are resources and the row underneath indicates the request types (HTTP GET, POST, PUT, or DELETE).

Request Types:

- **GET**: **Description**: Returns the event with the given `eventId`.
  Produces: application/json
  HTTP status code: 200: OK - The request has been responded as expected.

5.7.1.3 Current Events

URI: `REST_URL/events/current`

Request Types:

- **GET**: **Description**: Returns a list of the ten newest events sorted by event start time in descending order.
  Produces: application/json
  HTTP status code: 200: OK - The request has been responded as expected.

5.7.2 Mapping Interface

Description: The interface to access the location data of athletes or the event track.

URI: `REST_URL/map`

Output: The produced JSON object of a coordinates holds the following attributes with the datatype in parenthesis:

- `lat`: Latitude of the coordinate (number).
- `lng`: Longitude of the coordinate (number).
- `ts`: Creation date of the position (number).

5.7.2.1 Mapping Type

URI: `REST_URL/map/type`

Request Types:

- **GET**: **Description**: Returns all possible mapping algorithm to map the coordinates of an athlete (see Section 4.4).
  Produces: application/json
  HTTP status code: 200: OK - The request has been responded as expected.

5.7.2.2 Event Track

URI: `REST_URL/map/{eventId}`

Request Types:

- **GET**: **Description**: Returns a list of all racetrack coordinates for the event with the given `eventId`.
  Produces: application/json
  HTTP status code: 200: OK - The request has been responded as expected.

5.7.2.3 Event Track Mapped

URI: `REST_URL/map/{eventId}/{type}`

Request Types:

- **GET**: **Description**: Returns a list of all mapped racetrack coordinates for the event with the given `eventId`, using the given mapping type (`type`).
  Produces: application/json
  HTTP status code: 200: OK - The request has been responded as expected.

5.7.2.4 Athlete Track

URI: `REST_URL/map/athletes/{athleteId}/`

Request Types:

- **GET**: **Description**: Returns a list of all GPS coordinates for the athlete with the given `athleteId`, sorted by creation date of the GPS points in ascending order.
  Produces: application/json
  HTTP status code: 200: OK - The request has been responded as expected.

5.7.2.5 Athlete Track Mapped

URI: `REST_URL/map/athletes/{athleteId}/{type}`

Request Types:

- **GET**: **Description**: Returns a list of all GPS coordinates for the athlete with the given `athleteId`, using the given mapping type. The list is sorted by creation date of the GPS points in ascending order.
  Produces: application/json
  HTTP status code: 200: OK - The request has been responded as expected.

5.7.3 Chart Interface

Description: The interface to request the chart for athletes or the racetrack.

URI: `REST_URL/charts`

Output: The produced JSON object for a holds the following attributes (see Section 4.5) with the datatype in parenthesis:

- `type`: The chart type which is created. (string).
- `options`: The styling options of the chart (object).
- `data`: Contains the column descriptions and the data (object).

5.7.3.1 Chart Type

URI: `REST_URL/charts/type`

Request Types:

- **GET**: **Description**: Returns all possible charts for the athlete (see Section 4.5) or the event track.
  Produces: application/json
  HTTP status code: 200: OK - The request has been responded as expected.

5.7.3.2 Athlete Chart

URI: `REST_URL/charts/{id}/{type}`

Request Types:

- **GET**: **Description**: Returns a specific chart (`type`) for the athlete with the given `id`.
  Produces: application/json
  HTTP status code: 200: OK - The request has been responded as expected.

5.7.3.3 Event Track Altitude Profile

URI: `REST_URL/charts/{eventId}/altitude`

Request Types:

- **GET**: **Description**: Returns the altitude profile of racetrack for the event with the given `eventId`).
  Produces: application/json
  HTTP status code: 200: OK - The request has been responded as expected.

5.7.4 Statistic Interface

Description: The interface to request the chart for athletes or the racetrack.

URI: `REST_URL/stats`

Output: The produced JSON object holds a key-value pair with the static type as key and the calculation result as value.

5.7.4.1 Statistic Types

URI: `REST_URL/stats/type`

Request Types:

- **GET**: **Description**: Returns all possible statistic types for the athlete (see Section 4.3).
  Produces: application/json
  HTTP status code: 200: OK - The request has been responded as expected.

5.7.4.2 All Athlete Statistics

URI: `REST_URL/stats/{athleteId}`

Request Types:

- **GET**: **Description**: Returns all statistics for an athlete with the given `athleteId`.
  Produces: application/json
  HTTP status code: 200: OK - The request has been responded as expected.

5.7.4.3 Athlete Statistic

URI: `REST_URL/stats/{athleteId}/{statisticType}`

Request Types:

- **GET**: **Description**: Returns the requested statistic (`statisticType`) for an athlete with the given `athleteId`.
  Produces: application/json
  HTTP status code: 200: OK - The request has been responded as expected.

5.7.4.4 All Athletes Statistics for Event

URI: `REST_URL/stats/all/{eventId}`

Request Types:

- **GET**: **Description**: Returns a list of all statistics for every athlete participating in a event with the given `eventId`.
  Produces: application/json
  HTTP status code: 200: OK - The request has been responded as expected.

5.7.5 Upload Interface

URI: `REST_URL/gpx`

Request Types:

- **POST**: **Description**: Interface to upload one or multiple GPX files for an event. The interface also restores the missing elevation data of coordinates. Every time a client makes a request, all GPX files for the event are deleted.
  Consumes: multipart/form-data
  Form Parameters:
    `file`: File stream to access the GPX file.
    `eventId`: The event ID.
    `ukey`: An upload key to identify the uploaded files
  HTTP status code: 200: OK - The request has been responded as expected.

Chapter 6

Client

The client in the client-server architecture is a web application consisting of hypertext markup language (HTML), cascading style sheets (CSS) and JavaScript files. It is responsible for handling the interactions of the user such as clicking on buttons, links, or drag and drop elements. The server provides the data for the user interface (UI) via the representational state transfer (REST) interface as described in Chapter 5. Implementing the REST application programming interface (API), which provides the data for the UI, offers a natural separation of the data and the user interface. As stated before, creating the software architecture for this project follows the separations of concerns principle. For this reason and the implementation of a REST application programming interface (API), the client uses a model-view-controller (MVC) pattern in most cases. In 1979, abd Mikjel H [1979] introduced the MVC concept for the first time. The pattern separates the client in three components [Wesley 2012]:

- A model which provides the data. The data can be static from a text or JavaScript object notation (JSON) file, or dynamic, created from the REST interface.
- A view is a template which displays the data (HTML and CSS files). It is the representation layer of the MVC pattern. The view consists of elements which the user can interact with.
- A controller, which interacts with the view and the model. It is responsible to load data from the model into the view and handles the users interactions of the view.

Figure 6.1 shows the interactions between the components of the MVC pattern.

Additional to the MVC pattern, some elements of the UI use the model-view-viewmodel (MVVM) pattern. This pattern is a specialization of the presentation model invented by Martin Fowler [Carcangiu et al. 2016]. John Gossman modified the presentation model and published the MVVM pattern in 2005 [Syromiatnikov and Weyns 2014]. The MVVM consists of three components:

- A model which holds all the data, as described above.
- A view which displays the data from the model, as described above.
- A viewmodel which includes the UI logic. It is a mediator between the view and the model. The viewmodel sends and receives data from the model and automatically updates the view, which offers certain methods and services. It is the task of the viewmodel to handle these functionalities, for example, UI elements of the view that can trigger events. Another part of the viewmodel is to act as a binder. This informs the viewmodel about changes in the view or by a state change. It also converts data types of the model so that they can be used by the view.
  The used UI elements can bind custom functions to certain events that change the view. An example is changing the order of rows in a table, which executes the bind function and returns the new order of the rows.

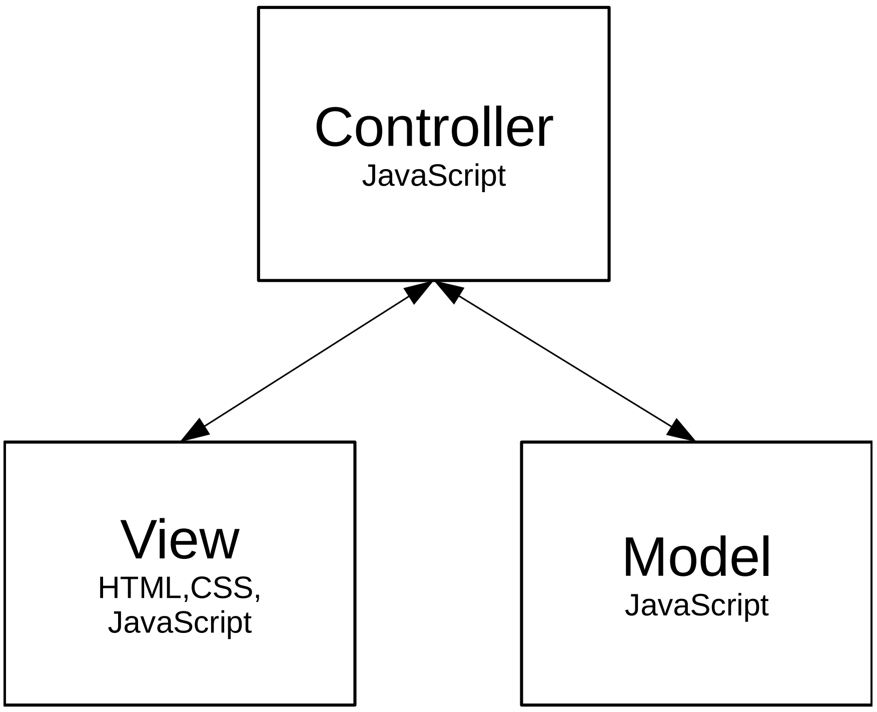


Figure 6.1: The MVC pattern.

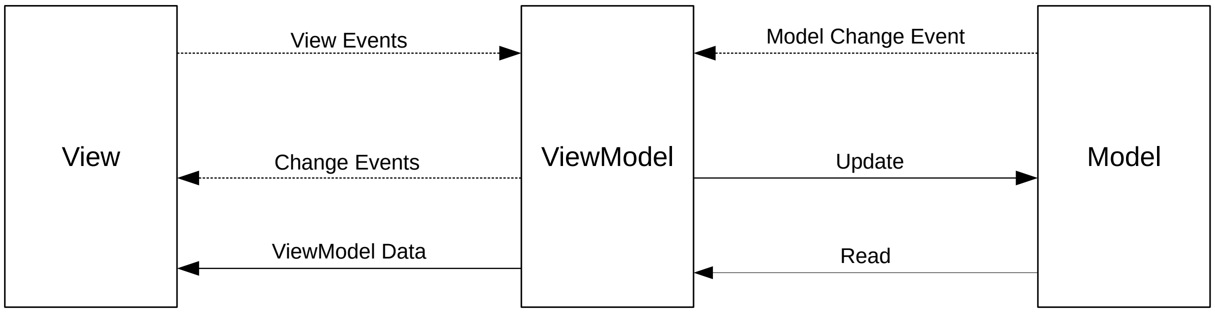


Figure 6.2: The MVVM pattern.

The interaction between the three components of the MVVM shows Figure 6.2.

Unless the UI does not use a predefined component which offers the MVVM pattern, the controller handles all interactions between the view and the model. The MVC pattern is highly complex and may not be worth the effort for simple user interfaces. This may be the case for the starting page, but the controller is necessary for the athlete tracking page. Both views have common functionalities, therefore parts of the controller of the athlete tracking page are reused for the start page controller.

6.1 Model

As described above, the model is responsible to access the data. The data is not directly stored in the model of the web application and must access the REST API of the server first. To access the data from the REST interface, the model uses the asynchronous JavaScript and XML (AJAX) concept. Furthermore, the model does not store the received data because it is handed over using the asynchronous functions. Every component that accesses the model must handle the data management, which also follows the separation of concern software design principle. The web application is programmed in JavaScript, and because of that the AJAX functionality is already part of the JavaScript language. Using the default XMLHttpRequest provided by JavaScript does not include any error handling and does not automatically add the needed attributes to the request header. The jQuery library is already needed for the user interface manipulation and includes its own improved AJAX library. These AJAX functionalities build on the XMLHttpRequest, but makes it more readable and easier to handle for the developer. The JavaScript code to perform an AJAX request is `$.ajax({})`. The `$` sign is the globally defined jQuery object. The ajax method takes an object as a parameter. The model uses the following properties:

- `url`: The `url` parameter defines the Uniform Resource Identifier (URI) to the REST interface. The REST interface and the web application files are on the same server, therefore the schema (http or https), the host address and port will always be the same. For that reason, the `url` parameter only consists of the path to the REST interface. The other components are added automatically.
- `type`: The `type` parameter defines the Hypertext Transfer Protocol (HTTP) method used for the request. For example, GET, POST, or PUT are defined in Section 5.5.7.
- `success`: If the server handles a request successfully and returns a response without errors, in most cases the response has the HTTP status code 200, which is the function that is passed as the parameter is executed. The `success` parameter only takes a function as value which can take three arguments. The first parameter is the JavaScript object containing the data returned form the server, a string that holds a status massage used as a second argument. The third one is a `XMLHttpRequest` object, which is a created request sent to the server. The success parameter is very important because the function is executed asynchronously.
- `error`: If a request is not successful it executes the function bind to the `error` property. This function also takes up to three arguments. A XMLHttpRequest object is the first parameter, a string that includes a status message as a second parameter. The last argument is a string that contains the error message from the server. The errors can be distinguished into two categories: a client error or a server error. A client error is very unlikely because the AJAX calls are tailored to REST interface. In practice, the errors that are likely to occur are server errors, for example, a timeout error, or the service is not available due to a problem with the server. This argument is used to inform the user via an info window that an error has occurred.
- `data`: The data parameter offers the possibility to use a JavaScript Object as a query fragment of the URI, as described in Section 5.4. Before the client sends a request, the library converts the key-value pairs of the JavaScript object into a query fragment and automatically appends that string to the URI. For example, the variable {name: "Max", age: 30, residence:"Graz"} is converted into the query string `?name=Max&age=30&residence=Graz`. Every request that is performed from the model uses the query parameter to append the current used language. For the REST interface it does not matter if the URI includes a query fragment or not. Nevertheless, it still creates a response. Planning with foresight, every request adds the current used language of the user interface as a language code to the query component. This makes it easier for developers to enhance the functionalities of the REST interface regarding better language support.

To structure the implementation of the model more clearly, it is subdivided into multiple layers according to the different resources implemented by the REST interface, shown in Figure 5.2. Structuring the code in JavaScript is a tricky situation. Although all modern web browsers have JavaScript support, not every browser supports all features of the newest JavaScript version. For this reason, the design decision was to keep the implementation as simply as possible. To support a very large number of devices, only basic features of JavaScript are used. For example, using ECMAScript 6 (ES6) classes[1] to structure the

[1]Ecma, *ECMAScript 2018 Language Specification.* https://www.ecma-international.org/publications/files/ECMA-ST/Ecma-262.pdf (visited on 20/03/2019).

code would prevent 13% of the global devices from using the web application[2]. To accomplish a good structural code and support a high number of devices, the implementation uses the versatile characteristics of JavaScript objects. Variables in JavaScript do not have a specific type; a variable can be a string, a number, an array or another object. With these characteristics it is possible to emulate ES6 classes. The model is one big JavaScript object containing other objects, which are the different resources of the REST interface.

Listing 6.1 shows a truncated structure of the model using JavaScript. The variable RestCalls (see Line 1) is globally set and can be accessed anywhere within the web application. The RestCalls object consists of multiple other objects, which represent the different resources. Every object defines the sub-path (Lines 6, 12, 18, 24, and 30) to the resource and the methods that execute the AJAX calls. In Lines 8, 14, 20, 26, and Line 32 of Listing 6.1, the variable ajaxCall is a placeholder, which implements all possible AJAX calls for the given resource.

Listing 6.2 shows a short section of the implemented model which is responsible for receiving the athlete coordinates. The method in Line 7 for example, retrieves all possible mapping methods from the server that are listed in Section 4.4. Line 19 is responsible for getting the coordinates for an athlete of a specific mapping type. Both AJAX methods use jQuery as shown in Lines 8 and 20. They take a JavaScript object as an argument which holds all properties, as described above.
To make the model more flexible to changes, the url (Lines 9 and 21) of every request is a concatenation of the constant prefix RestCalls.url + this.path and a suffix which is the path to resource. The concatenation makes it easier to test the model. For example, the servers offer a test service that returns dummy data with the URL prefix `/rest/test`. The model only needs to change one line to use the test services. The suffix is either constant or a concatenation of constants and variables. Line 9 shows the case of a constant suffix path to get all mapping types. To receive all coordinates of an athlete, the suffix is a concatenation of the the constant athlete/ and the athleteID/, shown in Line 21 followed by the mapping type mapping. For example, an athlete named "FH Kärnten" with the identifier (ID) 335 attends the event Kosiak, the path to the resource using no mapping is rest/map/athletes/335/None.
Both methods (Lines 7 and 19) have a callback argument. The callback parameter is a function which executes asynchronously after the client successfully gets a response from the server. The function assigned to callback success property shown in Lines 11 and 23 are handed over from the controller. If the request is not successful, the routine calls the function assigned to error. The assigned function is called RestCalls.handleError and is globally defined and assigned to every error property of the AJAX call (Lines 14 and 26). The error type is handled differently depending on what type it is. If a timeout causes the error, the error handler tries to resend the request. In this case, the variable tryCount, see Lines 12 and 24 and the retryLimit, see Lines 25 and 25 are important. If a timeout error occurs, the error handler re-sends the same request and increments the tryCount variable. The error handler tries to re-send requests as long as it does not exceed the limit, or the server successfully sends a response. The maximum attempts are set to three and defined by the model, as shown in Line 3. In both methods, the query only contains the information of the language used by the client, see Lines 15 and 27.

Listing 6.3 contains a method to filter the events (Line 5). This function gets a JavaScript object called query as an argument. The object can contain multiple query parameters, for example, the start and end time to filter events or a string to search for a specific event. The possible query parameters are shown in Section 5.7. Line 6 adds the used language of the client to the query object to keep consistency throughout the implementation. After modifying the object, it is assigned to the data property (Line 10).

The controller or another code fragment can access the method of the Model by following the properties. For example, to execute the function getMappingTypes() (Line 7 in Listing 6.2) by using JavaScript code RestCalls.Map.getMappingTypes(callback); or RestCalls.Event.getEventsQuery(callback, query); to execute the

[2]Deveria, *Can I use. Support tables for HTML5, CSS3, etc.* `https://caniuse.com/` (visited on 20/03/2019).

```
var RestCalls = {
url: "/rest/",
retryLimit: 3,
...
Event = {
path: 'events/',
...
AjaxCalls
...
},
Chart = {
path: 'charts/',
...
AjaxCalls
...
},
Event = {
path: 'events/',
...
AjaxCalls
...
},
Map = {
path: 'map/',
...
AjaxCalls
...
},
Stats = {
path: 'stats/',
...
AjaxCalls
...
},
```

Listing 6.1: A truncated code of the model which shows the structuring JavaScript using JavaScript.

getMappingTypes() method (Line 5 in Listing 6.3). This approach to structure the code is also used by other implementations such as the controllers.

```
var RestCalls = {
        url: "/rest/",
        retryLimit: 3,

        Map: {
            path: 'map/',
            getMappingTypes: function (callback = null) {
                return $.ajax({
                    url: RestCalls.url + this.path + 'types/',
                    type: 'GET',
                    success: callback,
                    tryCount: 0,
                    retryLimit: RestCalls.retryLimit,
                    error: RestCalls.handleError,
                    data: addLangProperty()
                });
            },

            getAthleteTrackForEvent: function (eventId, athleteId, callback = null,
                type = '') {
                return $.ajax({
                  url: RestCalls.url + this.path + eventId + '/athlete/' + athleteId
                       + '/' + type,
                  type: 'GET',
                  success: callback,
                  tryCount: 0,
                  retryLimit: RestCalls.retryLimit,
                  error: RestCalls.handleError,
                  data: addLangProperty()
                });
            },
            ...
        }
}
```

Listing 6.2: Two examples for the implementation of methods using AJAX to receive data from the REST interface.

```
Event:
{
    path: 'events/',

    getEventsQuery: function (callback, query) {
        query.lang = "en-US";
        return $.ajax({
            url: RestCalls.url + this.path + 'query',
            type: 'GET',
            data: query,
            success: callback
        });
    }
    ...
}
```

Listing 6.3: An example of the implementation of an AJAX call with query parameters.

6.2 User Interface

The clients user interface (UI) uses Bootstrap (see Section 4.1.9) to create the layout. For the start and athlete tracking page, two layouts are available: a layout for mobile devices which has a screen resolution width that is smaller than 768 pixels and a layout for devices with a screen resolution width greater than 767 pixels. Depending on the clients devices, the layout will automatically change.

6.2.1 Used Layout Components

On the grid system (see Section 4.1.9) different components from Bootstrap framework are used. To use these components, the HTML element must use a specific CSS class in order to render the element accordingly.

6.2.1.1 Navigation Bar

The navigation bar is placed on top of the website and is used by the athlete tracking page. The navigation bar can hold various sub-components such as drop-down menus, texts, links, or forms. The navigation bar can collapse for devices with smaller screens.

6.2.1.2 Drop-Down Menu

A drop-down menu is a button which toggles an overlay by clicking on it. The overlay displays a list of links of text elements. The drop-down menus are used in the navigation bar of the athlete tracking page.

6.2.1.3 Modal

A Modal is a pop-up container within the web page. Modals are always in front of all other elements placed on the map. They consist of a header to set a title and a body to display the elements. Modals are used for adding and modify points of interest, displaying athletes and charts.

6.2.1.4 Carousel

A carousel is a slideshow, which automatically switches through the images. Every image of the slideshow can display a text in front of the image. It displays two arrows on the left and right side of the image to navigate back and forward. On the bottom there are indicators to display the current position. The start page uses the carousel to display events.

6.2.1.5 Alerts

Alerts are pop-up windows within the web page. They are used to inform users about certain events. The athlete tracking page uses alerts to inform the user if an athlete is near a point of interest.

6.2.1.6 Element Selector

For the element selection, a Bootstrap extension called bootstrap-select[3] is used. The extension adds a search box to the select form to search for entries and two buttons, which selects and deselects all elements. It shows all elements in a drop-down list and it is possible to select multiple elements. The selector is used for the language selection, athlete visibility selection and event selection of the upload page.

[3] SnapAppointments, *bootstrap-select*. https://developer.snapappointments.com/bootstrap-select/ (visited on 20/03/2019).

6.2.1.7 Bootstrap Table

Bootstrap Table[4] is a Bootstrap extension which makes it possible to dynamically build a table. The extension provides many functionalities and styling options which can be changed using JavaScript. In addition to that, the extension also adds a toolbar on top of the table, which includes a search field and a control element to toggle the column visibility (see Figure 6.23). Other features are the drag and drop functionally to reorder the rows and the possibility to sort columns in ascending or descending order.

To create a dynamic table, the following four properties for every column are used:

1. title: The column heading, which is mandatory.
2. field: A row of the table is a JavaScript object with multiple attributes. The value of field is the key of the row, which holds the value for this column.
3. formatter: A custom formatting function that changes the representation of the data in a column.
4. sortable: By setting sortable to true, the user can click on a column header to sort the data of the table by ascending or descending order.

The table data is an array of JavaScript objects which are in the rows. A row has multiple keys that are the field names, and the values of the keys are shown in a table.

6.2.2 Start Page

The start page displays all past, live, or upcoming events (see Figure 6.3). On the upper half of the page there is a banner consisting of an image with a welcoming text. Below the image there are the recent events, as shown in Figure 6.3a. The bottom half of the page consists of a filterable table and a footer where the user can change the language (see Figure 6.3b).

6.2.2.1 Highlight Events

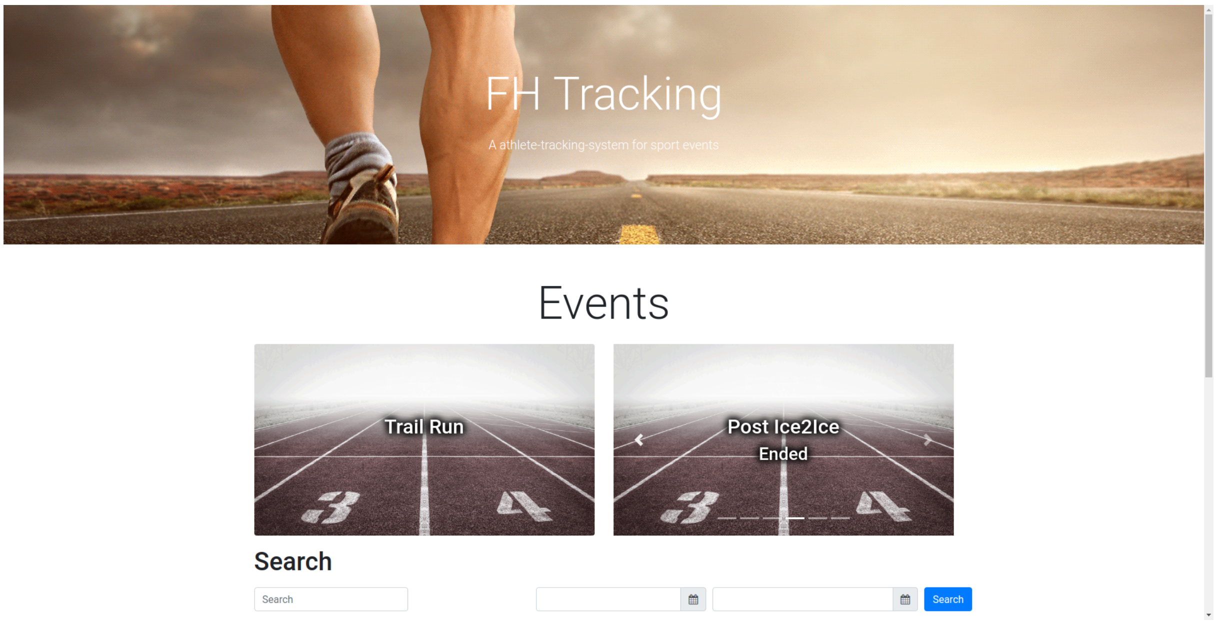
Carousels are used to display events. The carousel on the left side shows the live event and the one on right shows upcoming or past events, as seen in Figure 6.4. Figure 6.5 shows the two presentation types of the right carousel. Either an event starts in the future and displays a countdown (see Figure 6.5a) or has already finished (see Figure 6.5b). By clicking on the image, one can open the athlete tracking page.

6.2.2.2 Filter Events

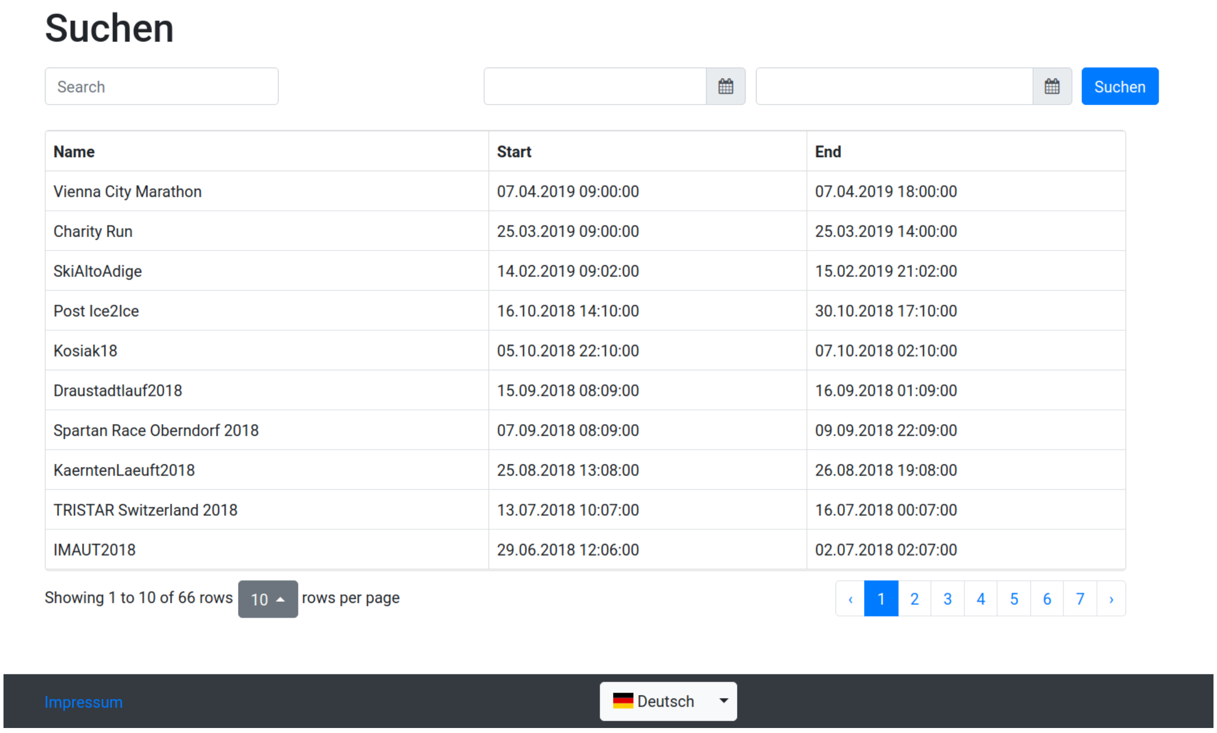
The user can display all stored events by using the table on the bottom of the start page, as shown in Figure 6.3b. To filter the table, two search parameters can be used: either searching for an event with a given name using the text field, or filtering the events inside a given time frame using the date picker. Combining the filtering options is also possible - Figure 6.8 shows the date selection form, which is displayed by clicking on the calendar icon. The user can choose a date using the calendar by clicking on date updates on the text field next to the calendar icon. The date format depends on the currently used language. To update the table, the search button on the right side must be clicked.

Below the table there are pagination options, as shown in Figure 6.6. It shows the total number of events and the number of visible ones. The user can change the number of table elements by using the drop-down button seen in Figure 6.7. To navigate between the table pages, the user clicks on the numbered buttons placed on the right side.

[4]bootstrap-table. `https://bootstrap-table.com/` (visited on 20/03/2019).



(a) The upper half of the start page displays the website banner and the carousels.



Name	Start	End
Vienna City Marathon	07.04.2019 09:00:00	07.04.2019 18:00:00
Charity Run	25.03.2019 09:00:00	25.03.2019 14:00:00
SkiAltoAdige	14.02.2019 09:02:00	15.02.2019 21:02:00
Post Ice2Ice	16.10.2018 14:10:00	30.10.2018 17:10:00
Kosiak18	05.10.2018 22:10:00	07.10.2018 02:10:00
Draustadtlauf2018	15.09.2018 08:09:00	16.09.2018 01:09:00
Spartan Race Oberndorf 2018	07.09.2018 08:09:00	09.09.2018 22:09:00
KaerntenLaeuft2018	25.08.2018 13:08:00	26.08.2018 19:08:00
TRISTAR Switzerland 2018	13.07.2018 10:07:00	16.07.2018 00:07:00
IMAUT2018	29.06.2018 12:06:00	02.07.2018 02:07:00

Showing 1 to 10 of 66 rows 10 rows per page

(b) The lower half of the start page displays the event table and the footer shows the web site footer.

Figure 6.3: The start page is split into two sections. (a) shows the upper half of the page and (b) the lower half. [Both screenshots taken by the author of this thesis.]

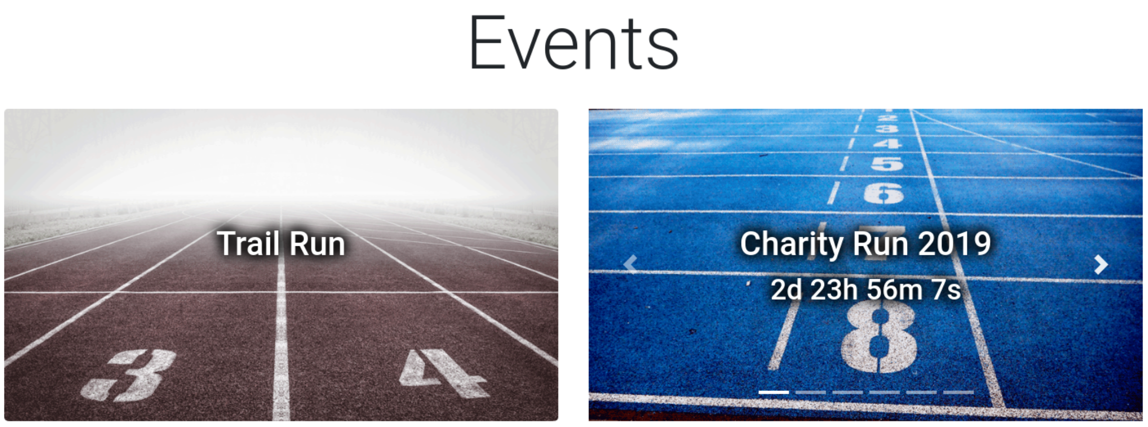
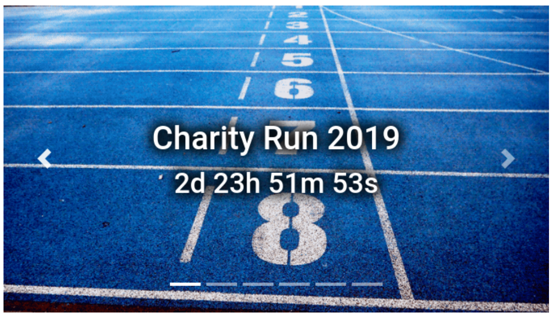
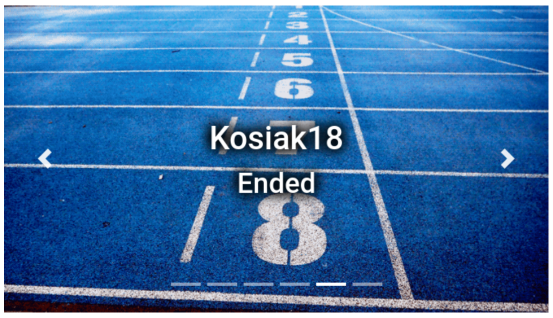


Figure 6.4: The two carousels placed on the website to highlight events.



(a) Upcoming events are displayed with a count-down timer as a text in the carousel.



(b) Past events are displayed with the text "Ended" in the carousel.

Figure 6.5: The carousel which shows the upcoming or past events are displayed in two different ways.(a) shows an upcoming event and (b) shows a past event. [Both screenshots taken by the author of this thesis.]

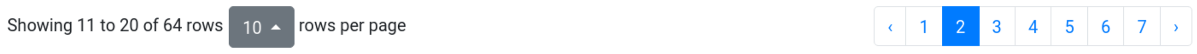


Figure 6.6: The pagination form where the user can navigate through the table entries.

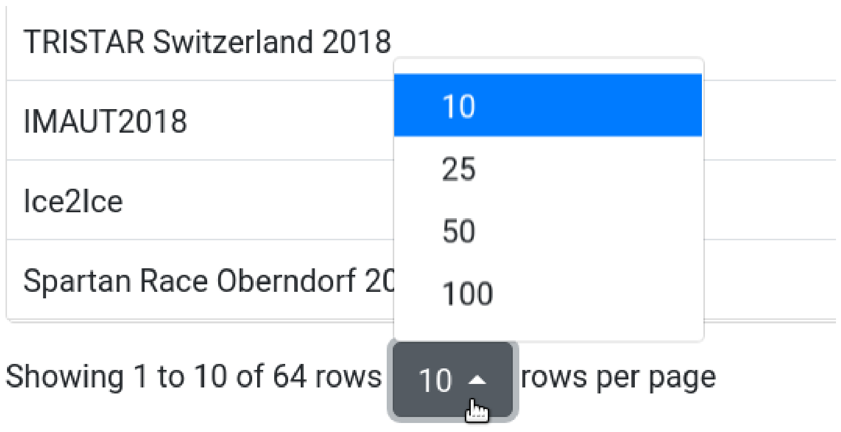


Figure 6.7: A drop-down menu where the user can change the number of rows in the table.

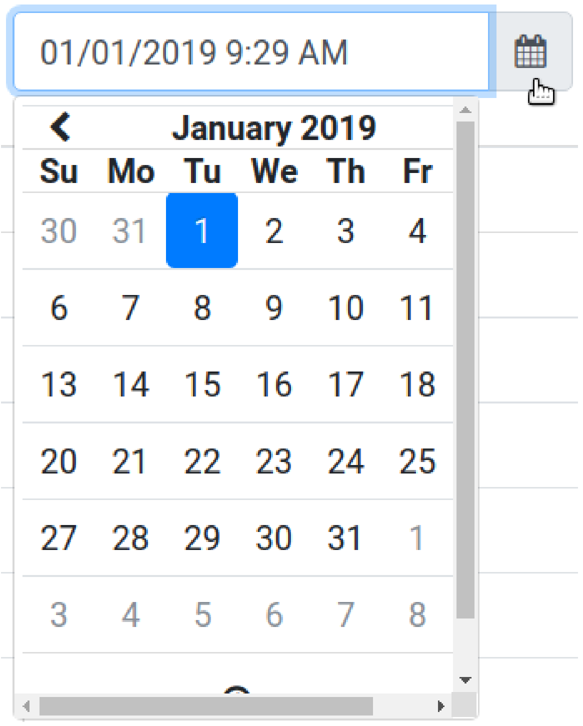


Figure 6.8: The date picker form where the user can select a date to filter the table.

6.2.2.3 Mobile Layout

The layout for mobile phones is more compact than the standard layout. Instead of placing the carousels side by side, the right carousel is pushed into the next row, as shown in Figure 6.9. The carousels' width is adjusted to the device as well as the event table, seen in Figure 6.10. All interactions with the items placed on the page are still possible using touch screens.

6.2.3 Athlete Tracking Page

Figure 6.11 shows the athlete tracking page for devices with a larger screen. The site consist of three elements: a navigation bar, a map and the athletes' statistics table.

6.2.3.1 Event Track Mapping and Information

If the user clicks on the button with the name "Track" placed in the navigation bar, a drop-down menu will open as shown in Figure 6.12. By clicking on the menu item, the "Altitude Profile" will display the chart in a modal window. Figure 6.12 shows the chart which is placed in the body of a modal. The "Info" button displays information about the event track and "Upload GPX" opens the upload page (see Section 6.3.6). The user can map the event track to a road using the implemented algorithm, as described in Section 4.4. The chosen mapping type is highlighted with a check mark.

6.2.3.2 Points of Interest (POIs)

Clicking on the "POI" button in the navigation opens a drop-down menu, as shown in Figure 6.14. The user can either add or edit points of interest. By clicking on the "Add" menu item, a modal opens in which the user can enter their name, position on the track and notification distance (see Figure 6.15). To add a POI to the map, the user clicks the "Add POI" button. All POIs can be shown by clicking on the "Show POI" menu item. This opens a modal that lists all the added POIs in the table, shown in Figure 6.16a. All entered values can be changed by clicking on them. Figure 6.16b shows the text field that appears

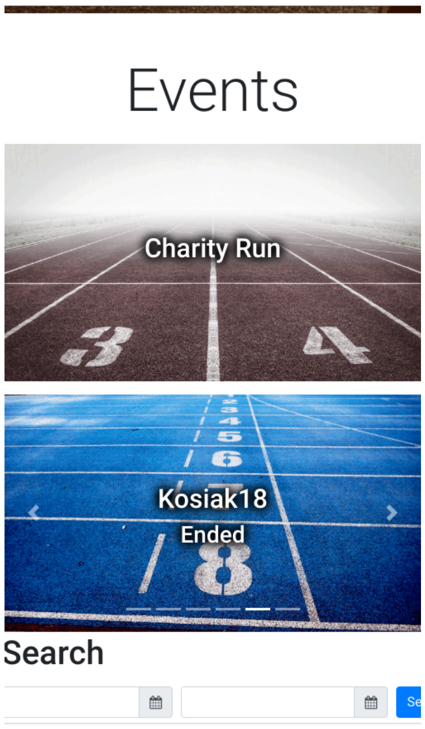
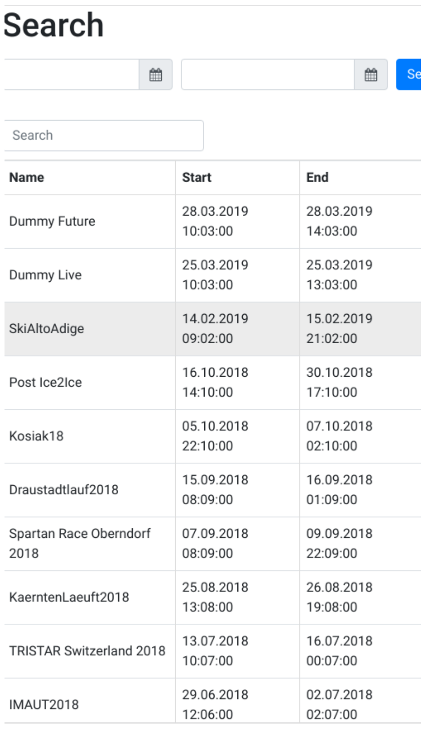
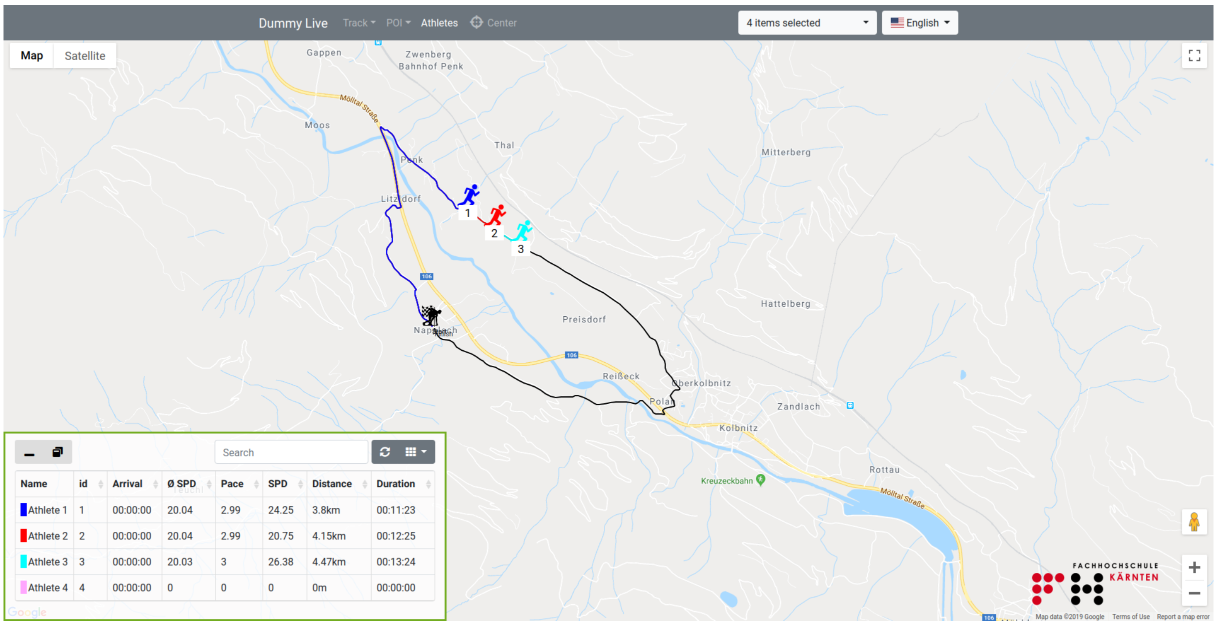


Figure 6.9: The mobile layout of the event carousels.



Name	Start	End
Dummy Future	28.03.2019 10:03:00	28.03.2019 14:03:00
Dummy Live	25.03.2019 10:03:00	25.03.2019 13:03:00
SkiAltoAdige	14.02.2019 09:02:00	15.02.2019 21:02:00
Post Ice2Ice	16.10.2018 14:10:00	30.10.2018 17:10:00
Kosiak18	05.10.2018 22:10:00	07.10.2018 02:10:00
Draustadtlauf2018	15.09.2018 08:09:00	16.09.2018 01:09:00
Spartan Race Oberndorf 2018	07.09.2018 08:09:00	09.09.2018 22:09:00
KaerntenLaeuft2018	25.08.2018 13:08:00	26.08.2018 19:08:00
TRISTAR Switzerland 2018	13.07.2018 10:07:00	16.07.2018 00:07:00
IMAUT2018	29.06.2018 12:06:00	02.07.2018 02:07:00

Figure 6.10: The mobile layout of the event table.



Name	id	Arrival	Ø SPD	Pace	SPD	Distance	Duration
🟦 Athlete 1	1	00:00:00	20.04	2.99	24.25	3.8km	00:11:23
🟥 Athlete 2	2	00:00:00	20.04	2.99	20.75	4.15km	00:12:25
🟦 Athlete 3	3	00:00:00	20.03	3	26.38	4.47km	00:13:24
🟪 Athlete 4	4	00:00:00	0	0	0	0m	00:00:00

Figure 6.11: The website for the athlete tracking.

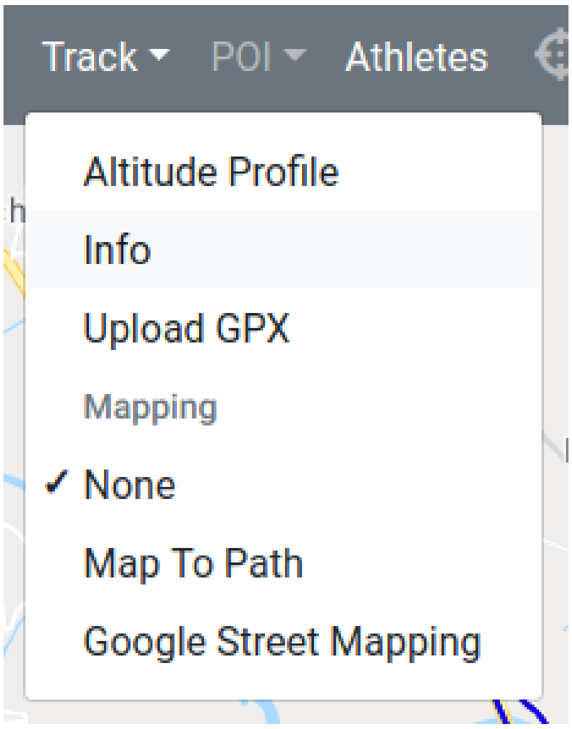


Figure 6.12: The drop-down menu of the "Track" navigation item.

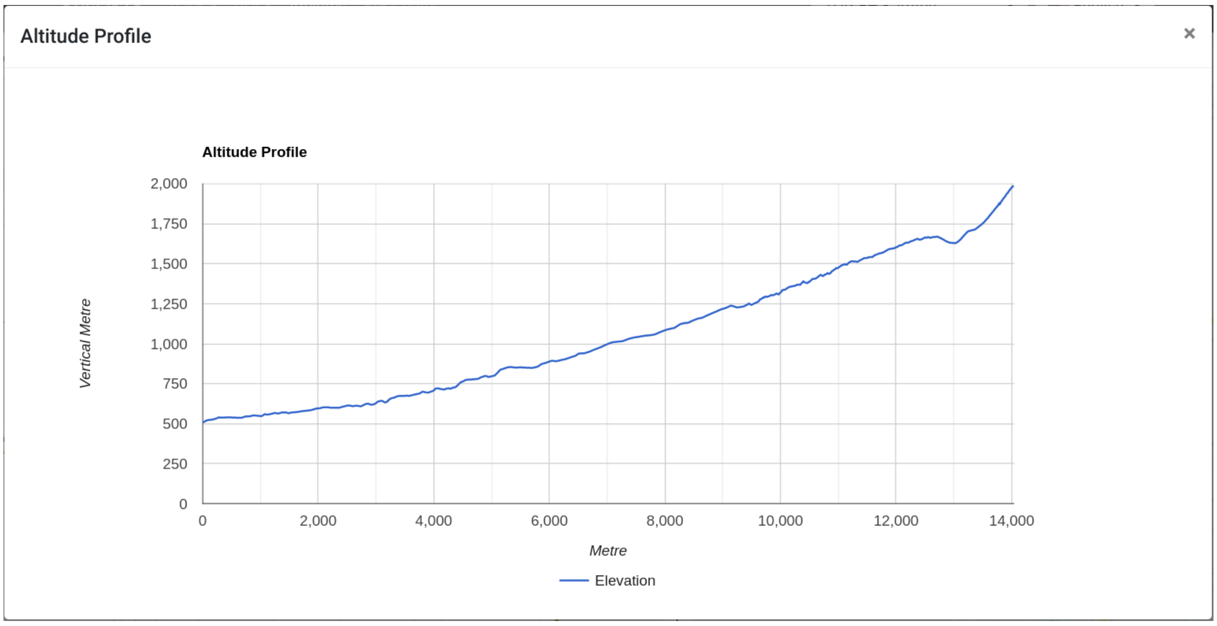


Figure 6.13: The modal which displays the altitude profile of an event track.

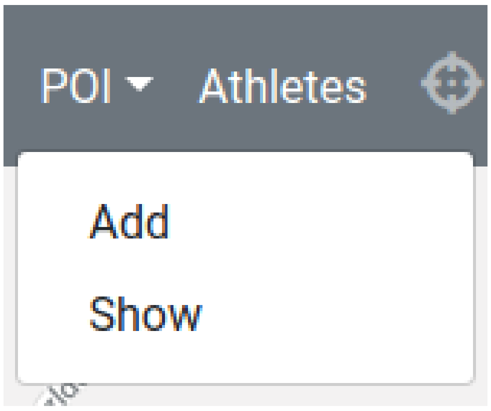


Figure 6.14: The drop-down menu of the "POI" navigation item

after the user clicks on an entry. The user can either save the changed value by clicking the button with the check mark, or cancel the modification by clicking on the "X". To remove a POI, the user must check the checkbox of the POI and click the "Remove" button. Figure 6.17 shows the placed POIs along the racetrack. If an athlete is within the set notification range, an alternate window pops up on the right side of the web page (see Figure 6.18).

6.2.3.3 Show Athletes and Change Mapping

Clicking on the "Athlete" button in the navigation bar will open a modal which displays additional information on the athletes, such as how long the GPS tracker is assigned to them. Figure 6.19a shows the modal which displays the athletes. The user can change the mapping type for every athlete, using the drop-down list on the right, seen in Figure 6.19b. Changing the mapping type will automatically update the athlete track with the selected mapping algorithm.

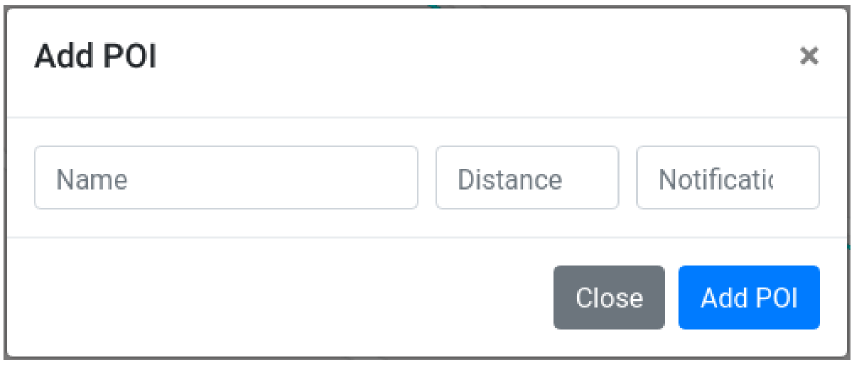
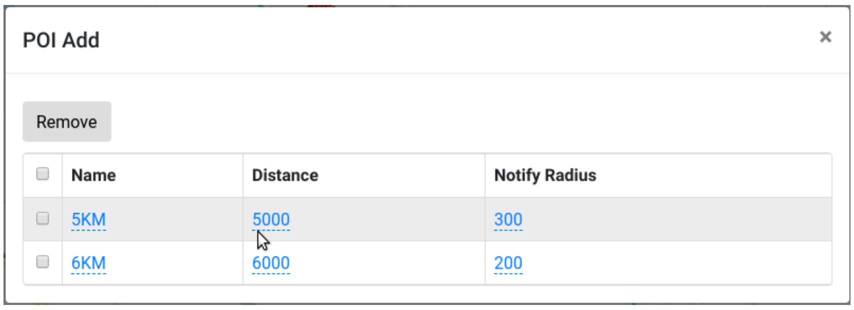
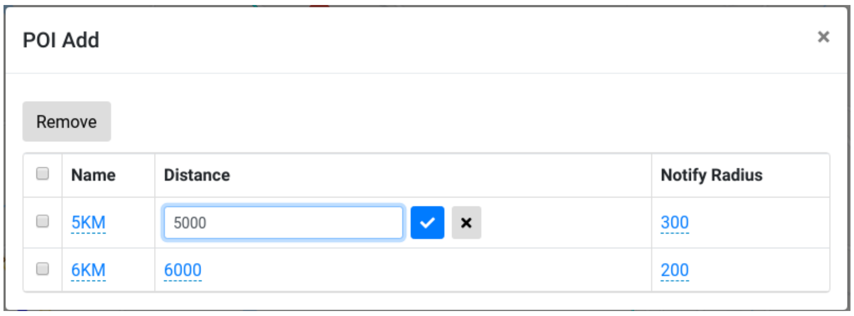


Figure 6.15: The modal where the user can add points of interest.



☐	Name	Distance	Notify Radius
☐	5KM	5000	300
☐	6KM	6000	200

(a) The modal which shows all points of interest.



☐	Name	Distance	Notify Radius
☐	5KM	5000 ✓ ×	300
☐	6KM	6000	200

(b) Table entry with enabled editing mode.

Figure 6.16: The editing modal for the points of interest where the user can show, edit, or remove a point of interest. (a) shows the initial state and (b) the editing mode. [Both screenshots taken by the author of this thesis.]

6.2.3.4 Change Athlete Visibility

The visibility of athletes can be changed using the selection drop-down window shown in Figure 6.20. The selector has a search field to filter athletes and two buttons to select or deselect all elements. The user can change the visibility of an athlete by clicking on their entry. Three visibility types are available:

1. An athlete is fully visible (route icon), which is referred to as the "fully visible" state. The poly-line and marker of an athlete are visible on the map.
2. Only the marker of an athlete is visible (marker icon), referred to as the "marker only" state.
3. The athlete is not visible. The poly-line and the marker are invisible, referred to as the "not visible" state.

Figure 6.21 shows an example of the different visibility types of the athlete displayed on the map. If

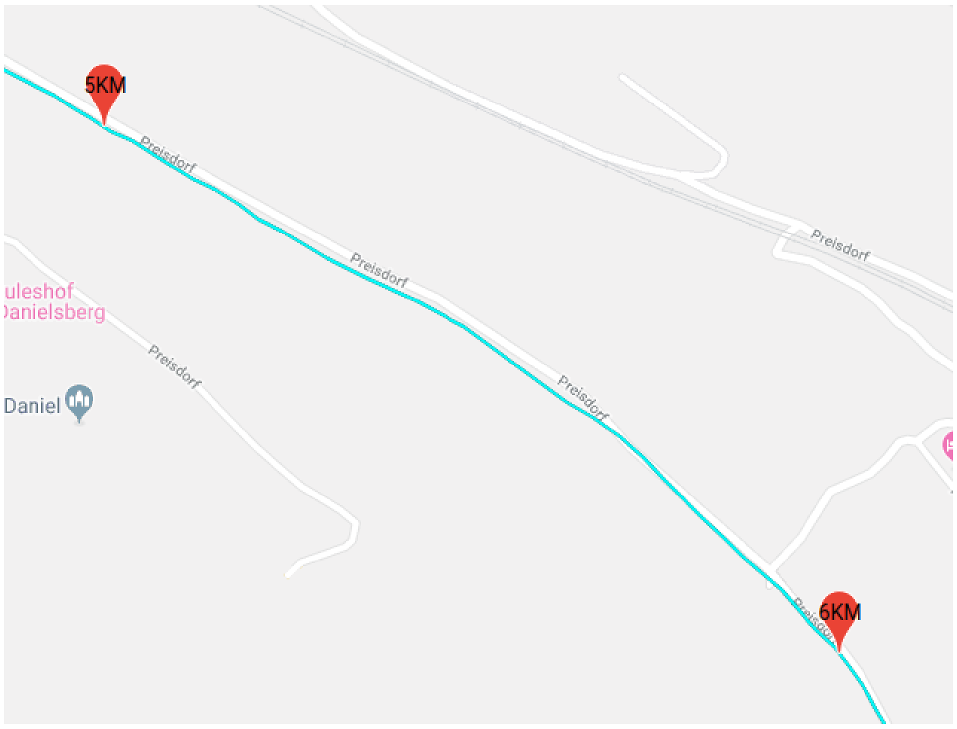
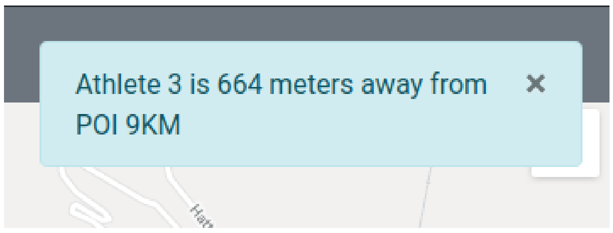


Figure 6.17: Two POI markers placed on the map.



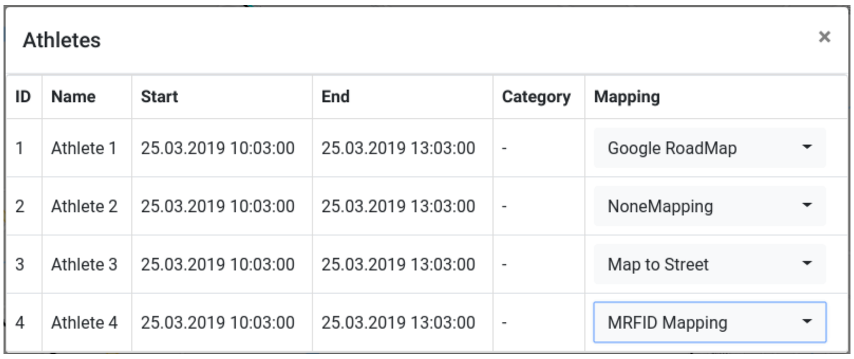
Athlete 3 is 664 meters away from POI 9KM

Figure 6.18: The notification alert that pops up if an athlete is near a point of interest.

an athlete attends an event, but the device has not recorded any GPS points, the athlete is grayed out.

6.2.3.5 Athletes' Statistics

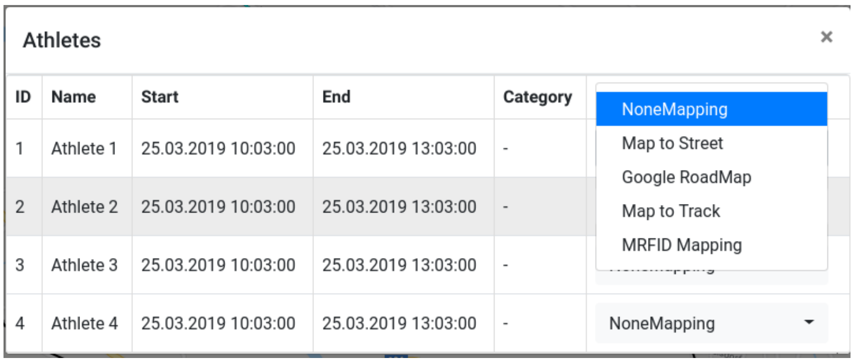
The athletes' statistics table is placed on the bottom left side of the athlete tracking page. It displays all available statistics for every athlete, shown in Figure 6.22a. Beside the athlete's name, there is a colored rectangle with the same color as the marker, and the poly-line of the athlete. This should make it easier to find athletes displayed on the map in the table and vice versa. The user can reorder the rows of the table by using the drag and drop feature. This will also change the drawing order of the athletes on the map, as described in Section 6.3.4.2. Figure 6.22b shows the dragging event of a row, which changes the cursor and the background color of the row. The user can also hide columns of the table by clicking on the toggle icon, which opens a window to show or hide the columns (see Figure 6.23). Figure 6.24 shows the table after deselecting the rows and two custom symbols are placed on the left side of the toolbar. The leftmost button minimizes the table, which will hide the table and show the two custom buttons on the bottom left side of the page, as shown in Figure 6.25. The minimize button is replaced with a maximized button, which can make the table visible again. The button on the right side of the minimize or maximize button is the pop-up button. Clicking this button will open the table in a new window and remove it from the athlete tracking page.



Athletes

ID	Name	Start	End	Category	Mapping
1	Athlete 1	25.03.2019 10:03:00	25.03.2019 13:03:00	-	Google RoadMap
2	Athlete 2	25.03.2019 10:03:00	25.03.2019 13:03:00	-	NoneMapping
3	Athlete 3	25.03.2019 10:03:00	25.03.2019 13:03:00	-	Map to Street
4	Athlete 4	25.03.2019 10:03:00	25.03.2019 13:03:00	-	MRFID Mapping

(a) The model that displays all the athletes.



Athletes

ID	Name	Start	End	Category
1	Athlete 1	25.03.2019 10:03:00	25.03.2019 13:03:00	-
2	Athlete 2	25.03.2019 10:03:00	25.03.2019 13:03:00	-
3	Athlete 3	25.03.2019 10:03:00	25.03.2019 13:03:00	-
4	Athlete 4	25.03.2019 10:03:00	25.03.2019 13:03:00	-	NoneMapping

(b) The opened drop-down list to change the mapping type.

Figure 6.19: The modal which displays all athletes where the user can modify the mapping. (a) shows the initial state. (b) shows the the drop-down list if the user wants to change the mapping type.

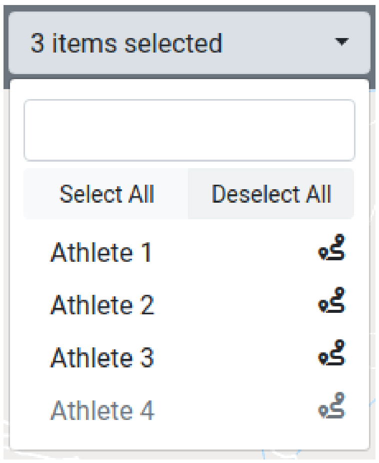


Figure 6.20: The athlete selector to change the visibility of athletes. "Athlete 4" is grayed out, because not GPS coordinates are recorded.

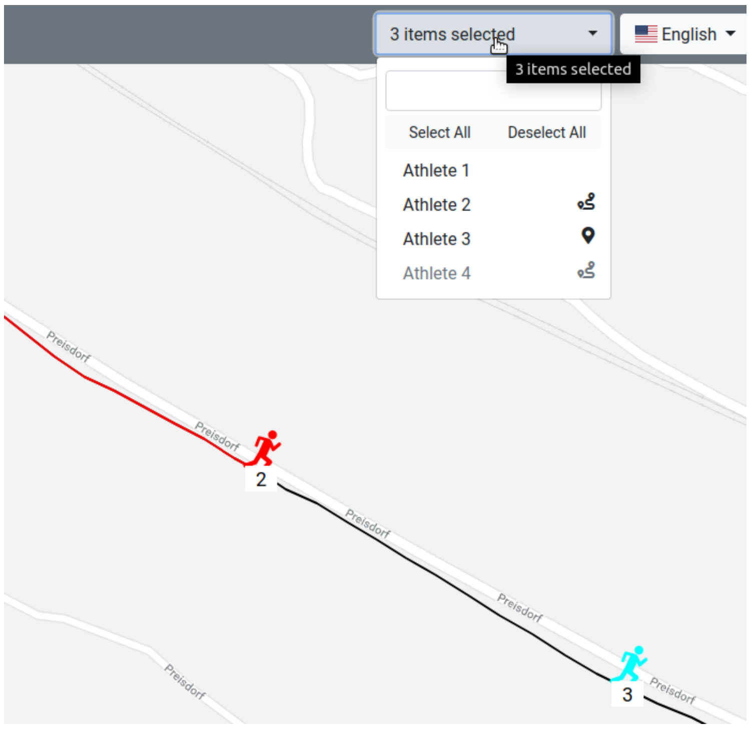
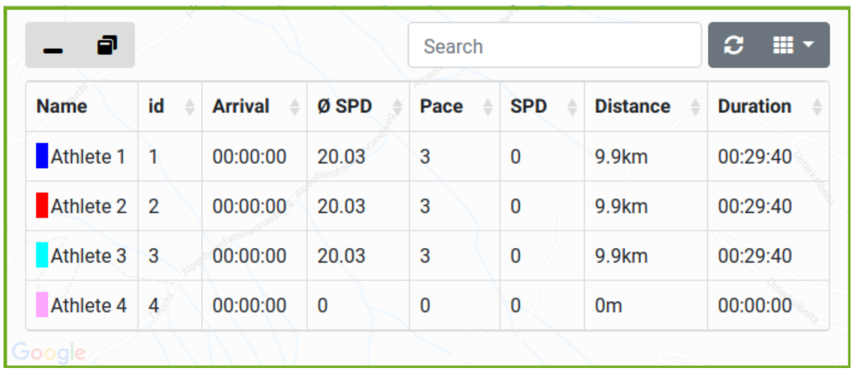
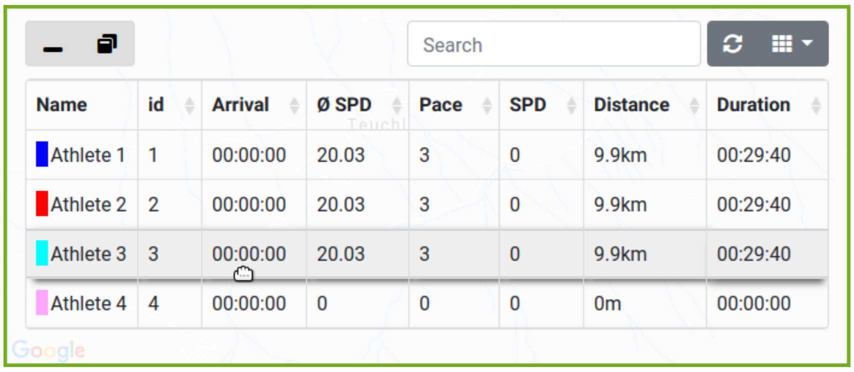


Figure 6.21: Changing the selection state in the drop-down list will change the athlete visibility on the map.



Name	id	Arrival	Ø SPD	Pace	SPD	Distance	Duration
🟦 Athlete 1	1	00:00:00	20.03	3	0	9.9km	00:29:40
🟥 Athlete 2	2	00:00:00	20.03	3	0	9.9km	00:29:40
🟦 Athlete 3	3	00:00:00	20.03	3	0	9.9km	00:29:40
🟪 Athlete 4	4	00:00:00	0	0	0	0m	00:00:00

(a) The athlete statics table.



Name	id	Arrival	Ø SPD	Pace	SPD	Distance	Duration
🟦 Athlete 1	1	00:00:00	20.03	3	0	9.9km	00:29:40
🟥 Athlete 2	2	00:00:00	20.03	3	0	9.9km	00:29:40
🟦 Athlete 3	3	00:00:00	20.03	3	0	9.9km	00:29:40
🟪 Athlete 4	4	00:00:00	0	0	0	0m	00:00:00

(b) The styling of the cursor and row changes, using the drag and drop feature to reorder the rows.

Figure 6.22: The athletes' statistics table shows all available statistics for the athletes (see Figure (a)). (b) shows the drag and drop event to reorder the rows. [Both screenshots taken by the author of this thesis.]

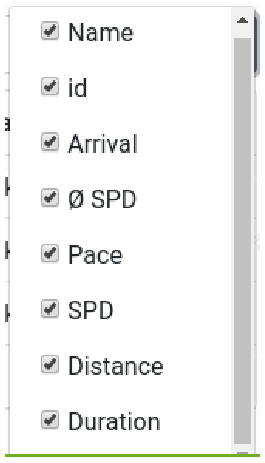
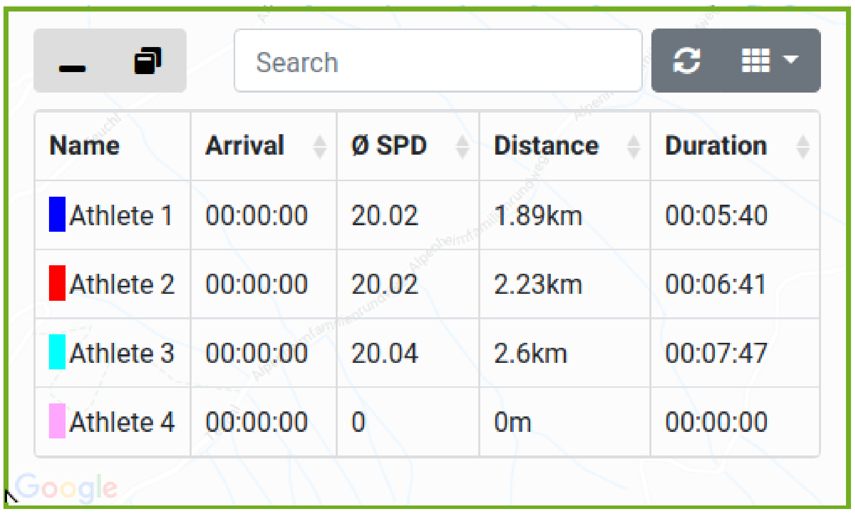


Figure 6.23: The window where the user can change the visibility of the columns.



Name	Arrival	Ø SPD	Distance	Duration
🟦 Athlete 1	00:00:00	20.02	1.89km	00:05:40
🟥 Athlete 2	00:00:00	20.02	2.23km	00:06:41
🟦 Athlete 3	00:00:00	20.04	2.6km	00:07:47
🟪 Athlete 4	00:00:00	0	0m	00:00:00

Figure 6.24: The athletes' statistics after using the toggle windows to hide columns.

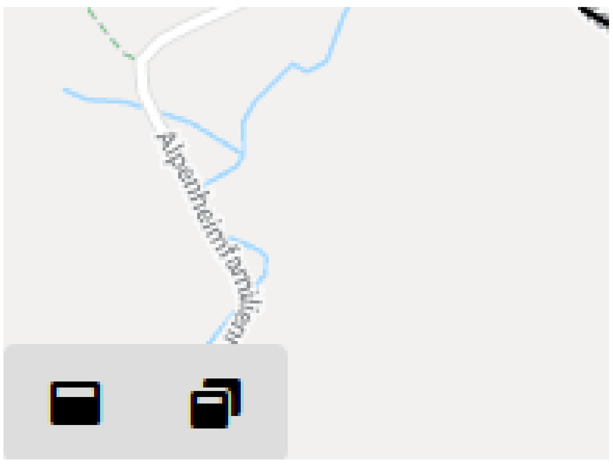


Figure 6.25: The buttons which are visible on the map, after clicking the "Minimize" button.

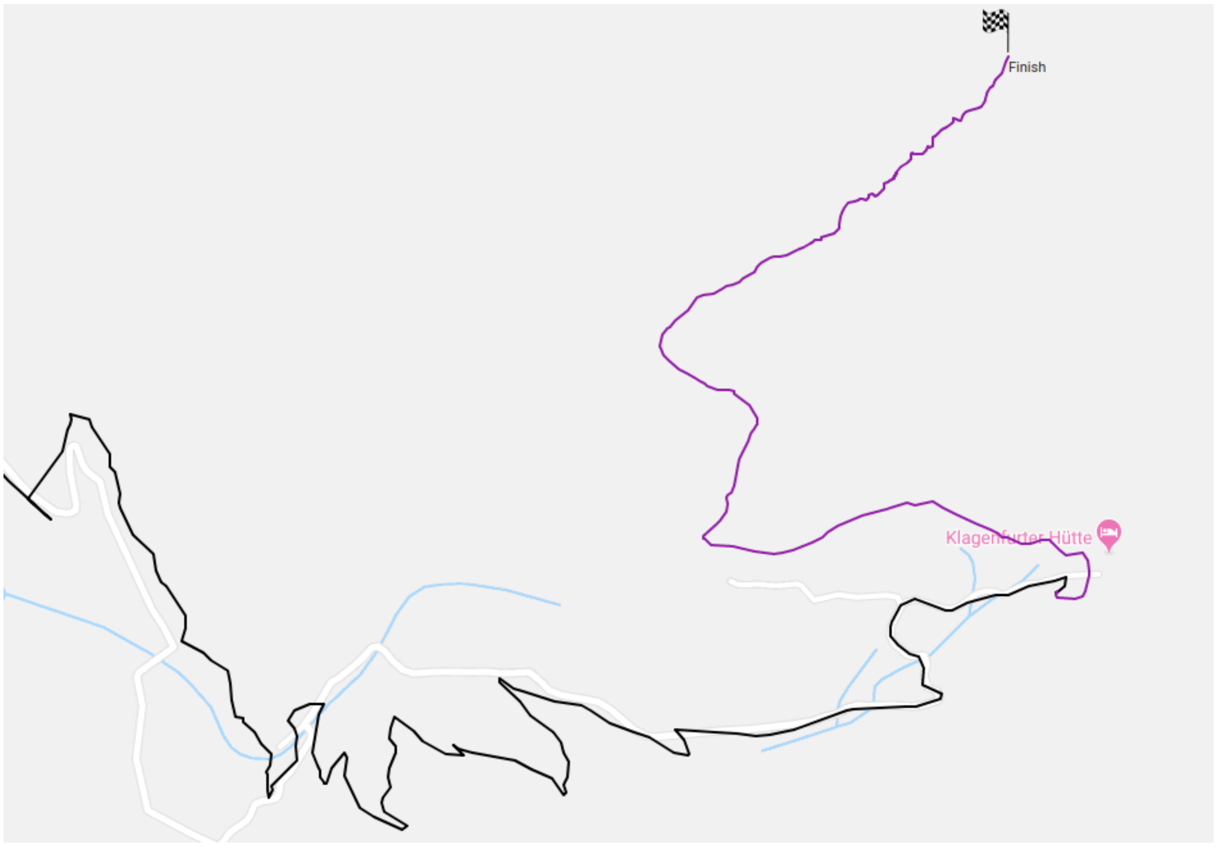


Figure 6.26: An event track displayed on the map consisting of two segments. The segments are varicolored to distinguish them.

6.2.3.6 Racetrack and Map

A racetrack can consist of one or multiple segments that are connected poly-lines on the map. Every segment has a different color to distinguish them easier. Figure 6.26 shows a racetrack which consist of multiple segments. The start symbol of the race track is a black runner's icon and the the finish symbol is a checkered flag, as shown in Figure 6.27.

An athlete consists of a poly-line and a runner icon as a marker, which uses a unique color for every athlete. Figure 6.28 shows an example of a colored athlete on the map. If the user clicks on the runner icon, an info windows pops up, seen in Figure 6.29. The info window displays all the possible charts that it is able to display. By clicking on a chart entry, this opens a modal which contains the rendered chart. Figure 6.30 shows a modal which displays the speed chart of an athlete and Figure 6.31 the pace chart.

6.2.3.7 Mobile Layout

Figure 6.32 shows the mobile layout for the athlete tracking page. The navigation bar is collapsed and the the statistics table can only be viewed in the pop-up windows by clicking the pop-up button placed on the bottom left of the screen. Clicking on the open icon on the top right of the screen opens the navigation window, as shown in Figure 6.33a. Clicking on the navigation item "Track" or "POI" shows the menu items. Figure 6.33b shows the sub-menu items of the "Track" navigation item. The modal size is adjusted to the smaller screen, as shown in Figure 6.34. All elements of the athlete tracking page, including the table, are also compatible with touch displays and do not limit any functions.

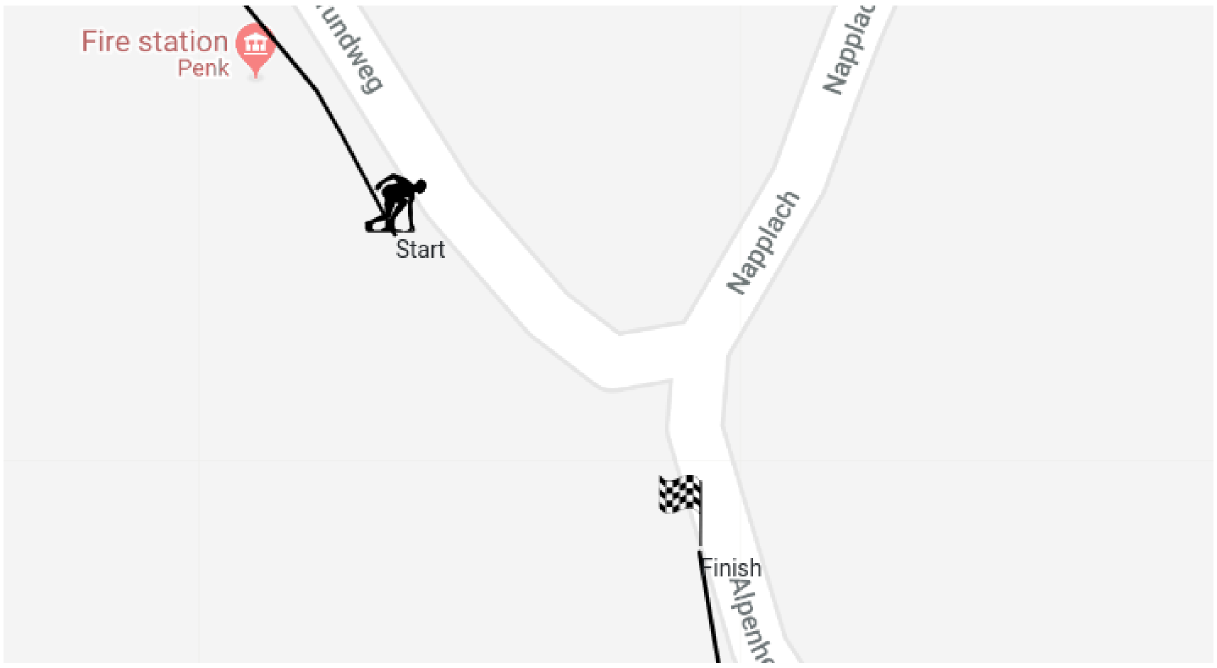


Figure 6.27: The start and finish markers of the race track.

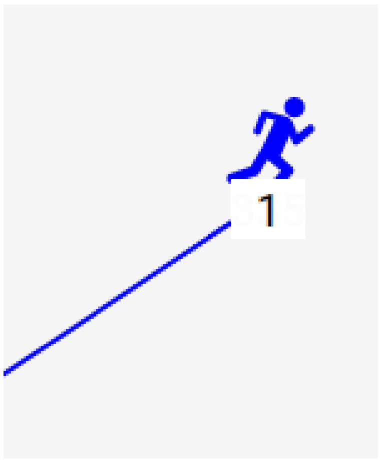


Figure 6.28: The athlete on the map consists of a marker and a poly-line, which have the same color.

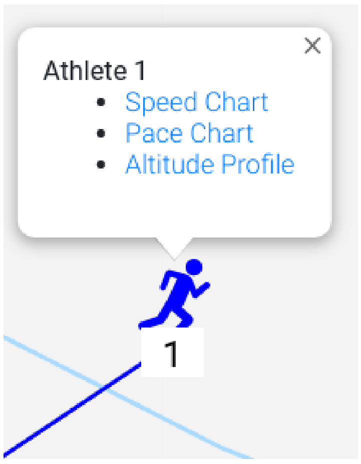


Figure 6.29: By clicking on an athlete to show the info window.

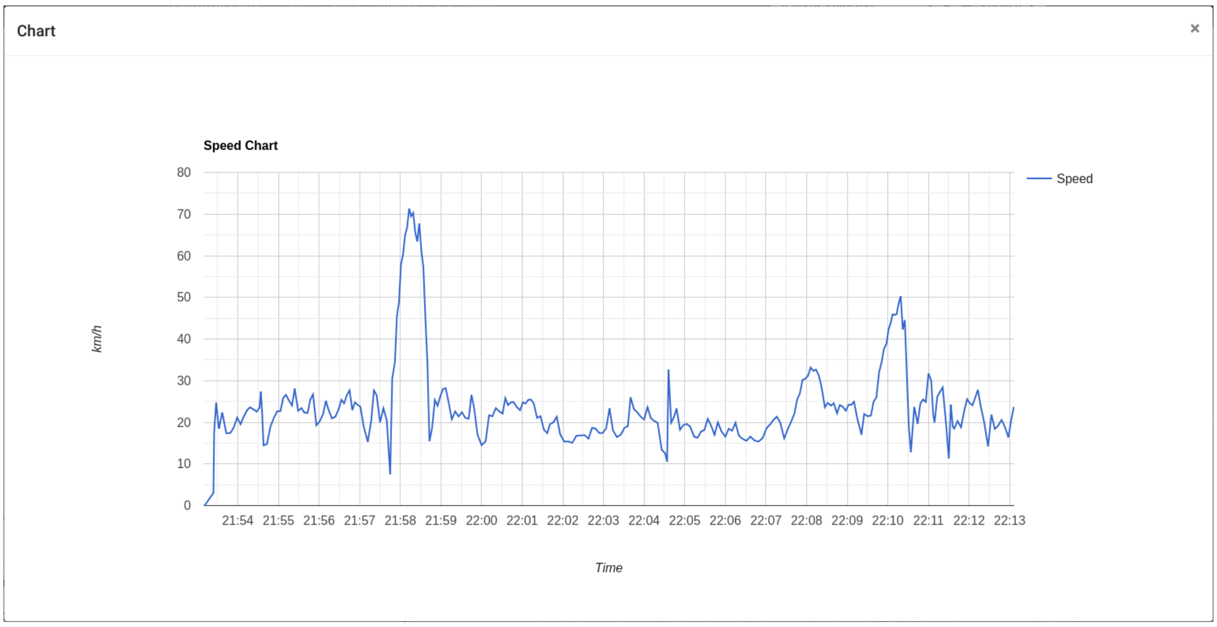


Figure 6.30: The modal which displays the speed chart of an athlete.

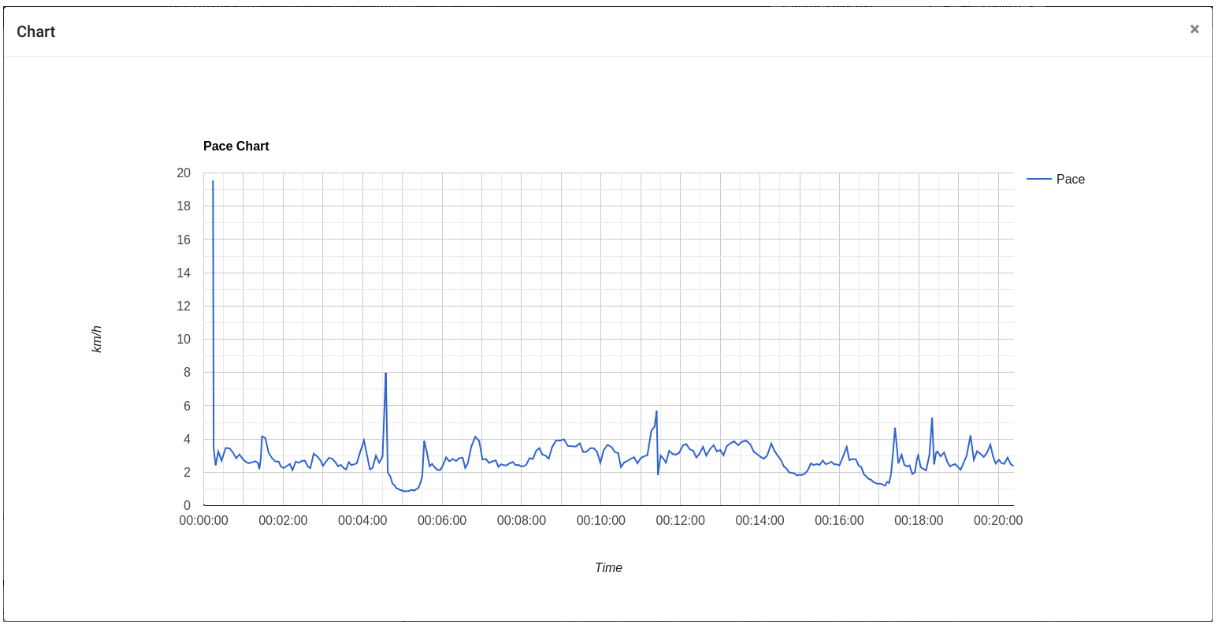


Figure 6.31: The modal which displays the speed chart of an athlete.

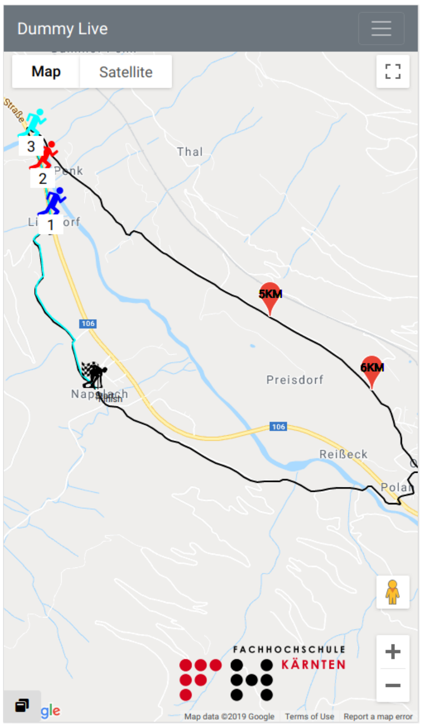
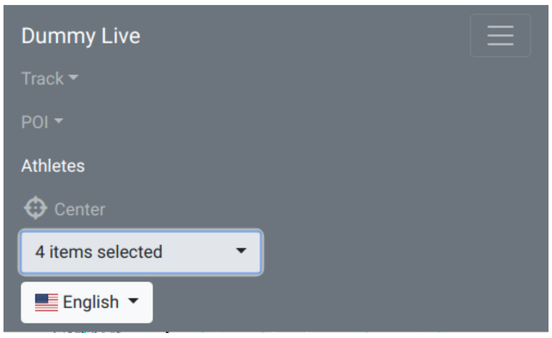
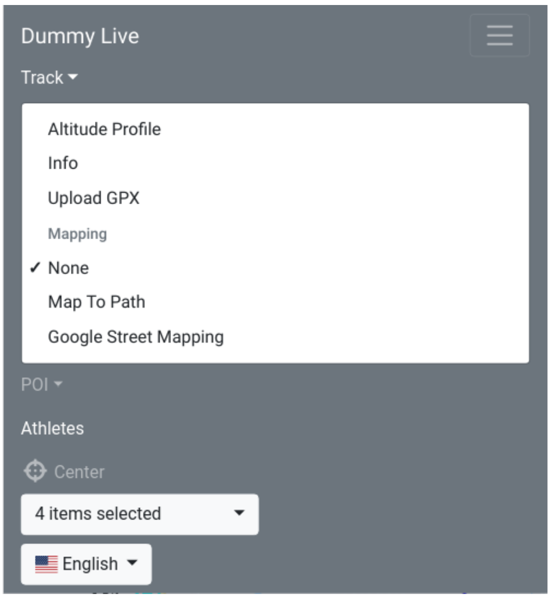


Figure 6.32: The mobile layout of the athlete tracking page.



(a) The navigation window of the mobile layout.



(b) The opened sub-menu of a navigation item.

Figure 6.33: The navigation window of the mobile layout. By clicking on a menu item which has sub-items, it will open a window as shown in (a). [Both screenshots taken by the author of this thesis.]

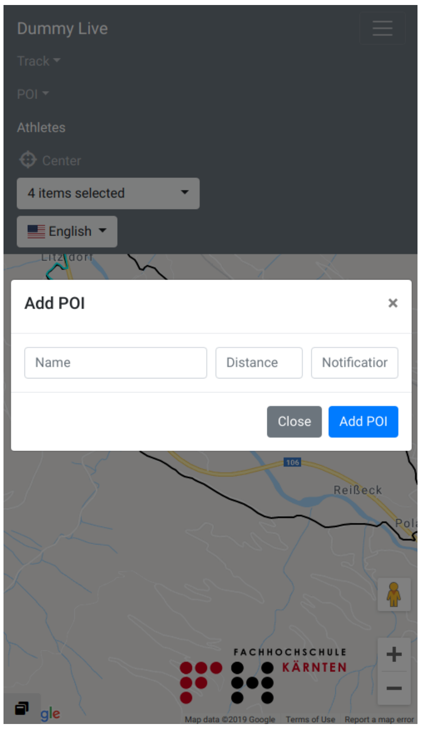
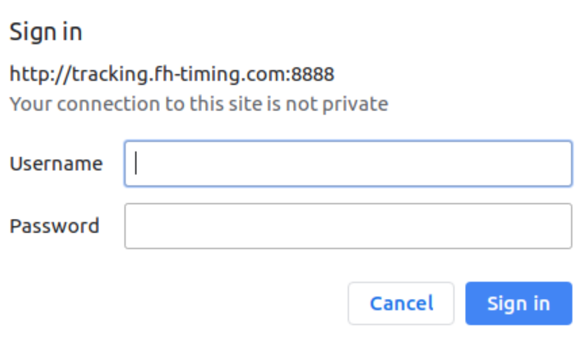


Figure 6.34: The mobile layout of a modal.


Sign in

http://tracking.fh-timing.com:8888

Your connection to this site is not private

Figure 6.35: The login form to access the upload page.

6.2.4 File Upload Page

Figure 6.36 shows the upload page to send racetrack files to the server. To upload a file, the client must select an event first by using the event selector. If the user accesses the upload page via the athlete tracking page, the selector is not visible. Clicking the green "Add Files" button will open the file explorer of the operating system, where the user can select the GPS exchange format (GPX) files. The selected files are shown in the list. The user can either remove all files using the "Cancel Upload" button, or a single file by clicking the "Cancel" button. By clicking the "Start Upload" button, it will upload the selected files to the server. To access the upload page, the user must enter their username and password in the login window, shown in 6.35.

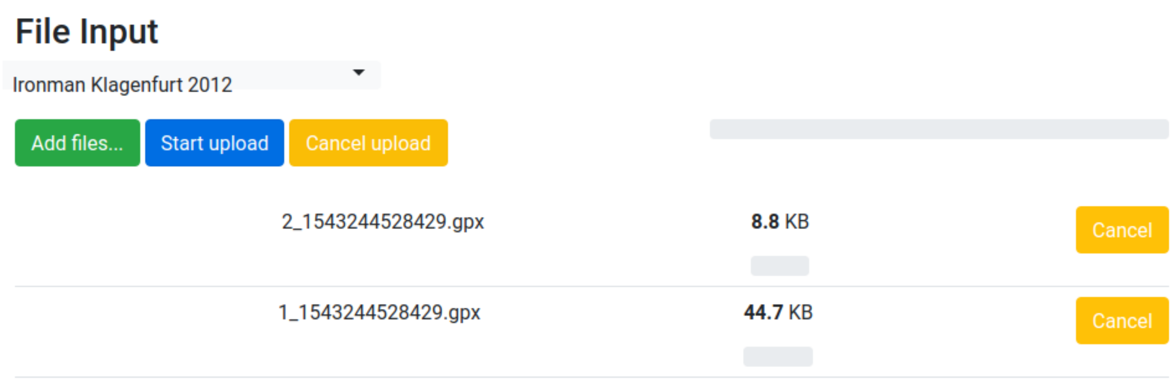


Figure 6.36: The upload page where the user can upload GPX files for an event.

6.3 Controller

The controller is the core part in the MVC pattern and it interacts with the model and the view. It is the most important part of this web application and it is responsible for many parts. Since the overhead is quite big, it is important to structure the controller well. This is accomplished by dividing it into multiple sub-controllers; each sub-controller is responsible for a specific group of task. As mentioned before, this separates the concerns in different controllers, which are much more lightweight than one big controller that handles everything. Another pro is that the code is much more manageable and easier to maintain, as splitting the controller in different components makes it easier to develop features independently. The splitting is possible because every controller can access the functionalities of the model to view elements without accessing other sub-controllers. The controllers are split into the following parts:

- A controller which handles the multilingual support. All sites of the web application use this feature.
- The table that displays the stats of every athlete has its own controller.
- A controller is responsible for creating the different charts for the athletes and the event track. This is only used by the athlete tracking app.
- Another controller is responsible for handling all interactions with the map.
- One controller is accountable for elements of the athlete tracking page. It manages all GUI elements except the map and the statistics table.

6.3.1 Asynchronous Functions

One of the main parts of the controller is to get data from the model, procedure them and change the view. Since the model does not store the data locally, it has to receive them from the server first. The internet connection of the client establishes the connection between the server and the client. Speaking from experience, internet connections are often not very reliable, especially when using smart phones using mobile internet. Depending on the internet connection of the client, the time between a request from the model and a server response may take several seconds. What may also occur is that the model has to resend the request because of a timeout error which multiplies the duration, as discussed in Section 6.1. In a normal application, which uses the MVC pattern, it is often directly connected to the database. It is because of this that there is little to no delay receiving the data. In this scenario, the program waits until the data is available to proceed. This is called the synchronous approach, where the data retrieving process blocks the rest of the program until the data is ready (assuming the program does not use threading for this process). JavaScript is single threaded, therefore the synchronous approach would block the whole

web application. For example, if a client performs many server requests by not using the asynchronous feature, it could block the whole application while waiting on the data, which decreases the usability and performance significantly.
Blocking the whole application by using synchronous code is not efficient, therefore the application uses the asynchronous feature. For instance, a code consists of two lines, named L1 and L2, where L1 requests data from the server. A synchronous program executes L1, waits until the server responds and then executes L2. In an asynchronous code however, the program does not wait for the line to finish. L1 makes an AJAX call and therefore L2 is performed without waiting for L1 to finish. L2 is executed without blocking because L1 is an asynchronous function. When the data is ready, the scheduler executes the function which processes the server response from L1. Let's assume the code has a third and fourth Line L3 and L4, which also executes asynchronous function calls. Since the program executes the L1 first and the L4 last, it does not conclude that the scheduler runs L1, L2 and L3 in the same order. The AJAX call which receives the data first also executes first. Figure 6.37 illustrates the difference between synchronous and asynchronous functions.

Peter Olson[5] brought up a good analogy for asynchronous tasks by comparing it with a restaurant with a single worker: people can sit down and order food, they do not have to wait for others to receive the food and finish eating. The order in which people receive their food depends on who long it takes to cook the food. The order in which people receive their food may not match the sequence people ordered the food in, but everyone gets their food as soon as it is prepared. The preparation time varies depending on what someone orders.

An important component using asynchronous functions are callbacks. A callback is a routine that is part of the asynchronous functions. The program executes an asynchronous function and when it is finished, the passed callback function is executed. Asynchronous functions, most of the time, are methods which process the data from external services such as Google API requests or accessing the REST API. Callback functions get the data via an argument to the callback from the asynchronous functions and process them. In some cases, a callback function needs more than one data source. Therefore, it depends on multiple asynchronous functions that collect the data. A common mistake made in this scenario is putting callbacks into other callbacks, which is commonly referred to as "callback hell". This creates a nested code that is not readable for other developers and hard to change or maintain. To avoid this problem, the nested function calls are split into different functions. If the first asynchronous function gets the data and executes the callback function, the callback function then executes the next function and hands over the collected data form the first asynchronous function call. This approach gives a clear structured code and makes the function call sequence more traceable.

6.3.2 Multilingual Support Controller

The multilingual controller is responsible for updating the language of the web page if the user wants to change it. The application uses the i18Next (see Section 4.1.10) multilingual software to solve this problem. As an addition to the base software, the controller uses a plugin for jQuery called i18Next[6]. The controllers' responsibilities are divided into two parts. First, the controller configures i18next and then it updates the user interface if the user wants to change the language. The controller does not use the REST API because the language files are accessible directly on the server such as an HTML or CSS file. To be able to use i18next the following configurations are set:

- The controller uses a language detection to obtain the used language of the web browser. If language detection is successful, the obtained language is used.

[5]Olson, *Introduction to Asynchronous JavaScript*. `https://www.pluralsight.com/guides/introduction-to-asynchronous-javascript` (visited on 20/03/2019).
[6]i18next, *plugin for jQuery*. `https://github.com/i18next/jquery-i18next` (visited on 20/03/2019).

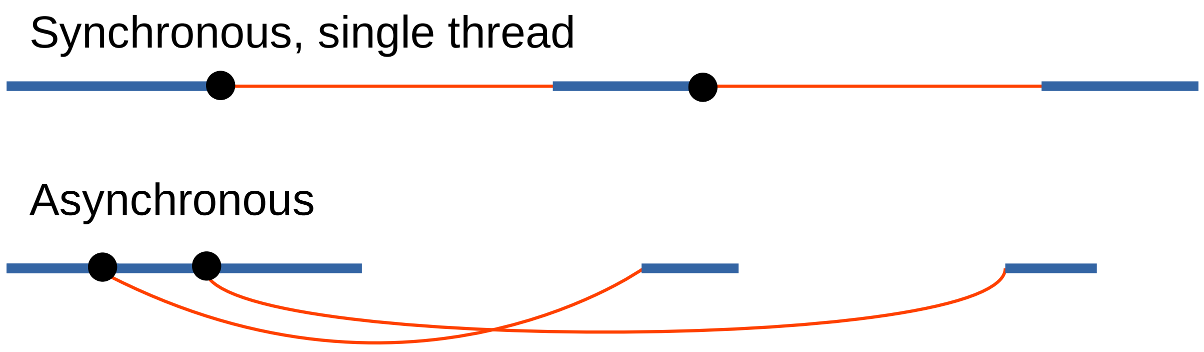


Figure 6.37: The figure shows the difference between a synchronous and an asynchronous function. Blue lines represent the execution of the code and red lines are the time the program has to wait for the server to receive the data.

- The location of the language files is in the same directory as the JavaScript files. A path must be set to access these files. This path uses a name convention to access the files such as ./locales/{{lng}}.json. {{lng}} is a wild card variable and it is replaced with the language code. The language code uses the RFC 5646 [Phillips and Davis 2009] standard and can either be a combination of language and region, or only the region. ISO 639-1[7] defines the language code standard and ISO 3166[8] defines the regional codes. For example, the code for Austria is de-AT and for Great Britain, en-GB. Language files can exists for a language and a specific region. For example, a language file called en.json contains the translation for English and a file named en-GB.json contains a specific translation for Great Britain. If a translation does not exist in a regional translation file, it uses the translation from the language file.
- The default fallback language is set to English. If the language detector selects a language where no translation is available, it uses fallback language. Additionally, if a language file misses a specific translation text, i18next uses the text from the fallback language.
- To improve the translation speed, it is possible to pre-load language files to the browser. If the user changes the language, the file is already available and does not need to be downloaded.
- The language controller stores the selected language configuration in the user's browser storage to improve the user experience. If the user visits the website again, it automatically selects the last used language.

After configuring i18next, it is now possible to use the library. If the user changes the language, the controller must update different sections and elements of the user interface. Using the jQuery extension of i18next makes it possible to translate certain areas of the website. To accomplish that, the controller selects certain areas using the jQuery selector. Containers which are automatically translated on every page are the navigation bar (`.navbar`) and modals (`.modal`). To translate such HTML containers, only two steps are necessary:

1. Change the language of the i18next object: `i18next.changeLanguage(newLanguage);`.
2. Select the container using the jQuery selector and translate it using the jQuery plugin. For example, to change the language of the navigation bar, the JavaScript code `$.('.navbar').localize();` is used.

[7]ISO, *ISO 639-1:2002 - Codes for the representation of names of languages*. https://www.iso.org/standard/22109.html (visited on 20/03/2019).

[8]ISO, *ISO 3166 Country Codes*. https://www.iso.org/iso-3166-country-codes.html (visited on 20/03/2019).

```
    <a class="nav-link dropdown-toggle" href="#" id="navbarDropDownTrack" data-
        toggle="dropdown"
    aria-haspopup="true" aria-expanded="false" data-i18n="map.nav.track.title">Track
        </a>.
```

Listing 6.4: Example of the usage of i18next in the navigation bar using the data-i18n attribute.

The localize() method function is like a tree walker algorithm: depending on the root element, the query selector returns zero or multiple elements. The localize() function parses every element and searches for the translation key. Only if an HTML element contains the attribute data-i18n, the function is able to set the corresponding translation text. Listing 6.4 shows the HTML tag of the navigation item "Track", which uses the data-i18n attribute.

The localize() function needs the value (map.nav.track.title) of the key data-i18n to replace the text "Track" with the current translation. Since the translation function of the jQuery plugin visits all elements recursively, it is important to select specific containers inside the document object model (DOM) and not just the root element (<body> tag). Using this approach makes the translation function more effective because the depth of the tree is not that deep, which decreases the computation costs and the amount of loops.

Not all elements of the web application are compatible with the i18next plugin. The controller must perform the translation process for tables such as the one which displays the statistics of athletes or shows all events differently.

As shown in Figure 6.3b, tables have predefined text elements. These elements are the paging indicator, the search box, the tooltip text of the buttons and a text placeholder when the table contains no data. The table already includes a translation for all these text elements. The controller has to execute the JavaScript code `$.("#table").bootstrapTable("changeLocale", "de-DE")` to change the language. The ID `#table` in the query selector is the table target to translate. "changeLocale" is the function of the table plugin to change to the new language ("de-DE").

The column descriptions are the second part of the table that needs to be translated. Every table on the web application that wants to translate the column descriptions must use the `data-i18nextPrefix` attribute in the HTML tag of the table. For example, the table which holds the statistics uses the following HTML code: `<table id="table" data-i18nextPrefix="map.stats"></table>`. The value of the attribute `data-i18nextPrefix` is the prefix used to get the key for the column heading translation. Each column of the table also stores a unique identifier. The key for the translation text of a column is the concatenation of the prefix and the ID of the column. For instance, the translation key for the column heading distance in the statistics table is map.stats.distance, where map.stats is the prefix and distance is the ID of the column. To translate all column headings, the controller has to build a JavaScript object with a key-value pair for each column. The key of an element is the ID of the column, and the value is the translation text. For instance, the JavaScript object can look like this: {"distance" : "Distanz", ... }. The key is obtained from the i18next object, which already includes the data from the translation files. To change the text of the column descriptions, the controller executes the following JavaScript code: `$.(#table).bootstrapTable("changeTitle", translationObject)`, where changeTitle is the function which replaces the column heading text with the values in the JavaScript object (translationObject).

Additionally, each site of the web application can add custom routines to the translation task. For example, by adding additional HTML containers that should be translated (used by the start page) or

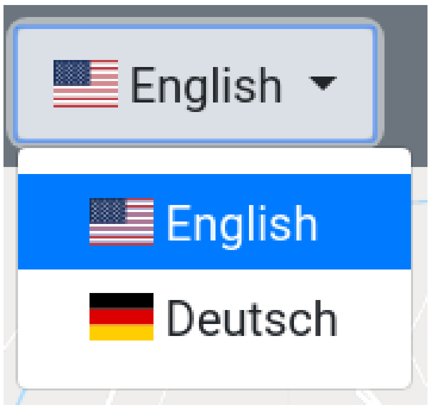


Figure 6.38: The language selector which is placed on the web page.

reloading specific scripts such as the athlete selector (see Figure 6.20). The athlete selector is a modified drop-down list with an integrated search bar and buttons to select or deselect all elements. The buttons and the search field include a text that is predefined inside the Bootstrap plugin. To change the text, the controller must reload the plugin with the correct language properties. This replaces the stored string in the code, but not in the user interface. Therefore, the controller has to recreate all selectors and restore all configurations such as elements, selection state and functions that are bound to specific events of the selector.

A user can change the language of a specific website using the language select form, as shown in Figure 6.38. The controller is responsible for creating the selector and translating the page if the user changes the language. If a new page wants to add multilingual support, it only takes three steps:

1. Create an HTML container with the ID languageContainer. The controller will automatically add the language selector inside this container. Figure 6.38 shows the language selector which is placed in the target container.
2. Add a link to the JavaScript file of the language controller to the website. After the web browser loads the document, the controller performs all tasks.
3. If needed, add some additional tasks to the translation process as described above.

6.3.3 Chart

The task of the chart controller is to use the produced data of the Rest API (see Section 5.7.3) and create a Google Chart object. In the current state of the application, only the athlete tracking page uses the chart controller. For this reason it would make sense to implement the chart logic into the controller which is responsible for the athlete tracking page. Although charts are only part of the athlete tracking page, there is no real dependency between the chart and the tracking web site. To generate a chart, only two pieces of information are needed: the data to create the chart and the target container (HTML tag), where the chart should be placed. Following the separation of concerns software design principle leads to the conclusion to separate the functionalities into a different controller.

To create a chart, an external controller binds the chart creation to an event of the user interface. For example, in the athlete tracking page, the user can click on an athlete and select a specific chart Type (see Section 4.5) to be displayed. The external controller binds this event to the creation function of the chart controller, which can either create a chart for the race track or a chart for an athlete. To create both charts, the controller needs an ID of the HTML tag. The controller will place the chart inside this container when they are able to render the chart. The altitude chart is the only available as a single option to create a chart for the racetrack. This option only needs the event ID in addition to the target container.

If the external controller wants to create an athlete chart, the athlete ID and the specific chart type (see Section 4.5) must be provided. With this information the controller can receive the data from the model.

The received data consists of three parts. The type of the chart, for example, `LineChart` or `BarChart`, the data set, which contains the data for the axis, and the options which are responsible for the styling and formatting of the chart. The controller needs these three variables and the target container ID to finally render the chart. To access the functionalities of the chart controller, an external producer uses the code `Chart.drawAltitudeChart(targetContainerID, eventID)`, or `Chart.drawAthleteChart(targetContainerID, athleteID, chartType)`. The globally defined variable `Chart` of the chart controller is accessible by any site including the JavaScript file.

6.3.4 Athlete Tracking Site Controller

The controller which is responsible for the athlete tracking is one of the core components in the whole web application. Since the athlete tracking page consists of multiple elements a user can interact with, it is useful to separate the functionalities into different sub-components. These sub-components are:

- Temporarily store and manage data for the map and athletes.
- Management of the map and the included elements.
- Creating the statistics table and handling user interaction.
- Dynamically add elements to the view and handling all user events, apart from the map and the statistics table. It is also responsible for updating the elements of the site.
- Store and restore user settings of the site to improve the user experience.

6.3.4.1 Data Storage

Since the model only passes the values to the asynchronous callback functions, it is important that some information be kept stored during a user session. These callback functions that process the data are also responsible for storing them. Therefore, the data storage does not need to make the same REST API calls to avoid unnecessary traffic. The data storage holds all the information about an event such as ID, name, start time, end time and description. The ID is especially important for all of the controllers to receive data using the model. Without the event and athlete's ID, it is not possible to get data from the model. The start and end time is essential for the controller to determine if an event is ongoing or not, which stops the update process.

The data storage also holds information about the map. Every element placed on the map stores a reference object. The objects on the map can be divided into three groups:

1. Point of interest (POI).
2. Elements on the map that are related to an athlete.
3. Elements that are related to the event track.

The storage container of the POIs is an array with POI elements. A POI element is a JavaScript Object which holds multiple properties such as the reference object of the marker that belongs to the POI and an ID. The name, distance on the event track and notified distance are also part of the element. Every POI element also stores the ID of the athletes that trigger a notification to avoid duplicate notification alerts.

The track object is much simpler, as it stores the two marker references for the start and finish line and an array for the poly-lines that are the racetrack and connects both markers.

Athletes are stored in a `Map` where the key is the athlete ID. Using an array for the athletes is not recommended. The IDs of athletes do not start at zero and may not be continuously numbered, therefore a JavaScript array would create empty entries. For example, if an event has two athletes with the ID 7 and 14, the array stores them at the index position 7 and 14. The array assigns empty entries to the indices 0 to 6 and 8 to 13. Using an array would produce unnecessary entries and the time to iterate such an array would not be efficient. The value of every key in the map is always a JavaScript object that contains the athlete's name and a reference object for the poly-line, marker, and info window. In addition to that it stores current mapping types and the visibility of an athlete. Both are important for the persistence of the website. It is important to store the reference objects, as without them it is not possible to change or delete them, for example.

6.3.4.2 Map Controller

The map controller is not a controller in the proper meaning of the word, regarding the MVC pattern. The controller creates the view and object on the map using the Google Maps JavaScript API and does not create any HTML elements. To create the map, Google Maps needs a target HTML container, equivalent to the charts creator. The script generates and loads all data via the Google servers and shows it within the target container. Google Maps JavaScript API only provides a limited amount of actions that can be performed on the map. Therefore, the view can only customize the location and the size of the HTML container, which contains the map.

The map controller is the central component of the whole application and the only part that directly changes the map. The first task of the controller is to create the map and place elements for athletes and the event track. Section 4.1.11 discusses the most important elements used by the map. The map controller requests the data for the event track and the athletes via the model. The event track consists of multiple elements, such as the start and finish line, and a path that connects them. Depending on the event, the path can consist of multiple sub-paths. For example, a duathlon has two sections, a running and a cycling segment, so the path that connects the start and the end point consists of two poly-lines. If the track has multiple segments, every poly-line has a different color, which makes it easier for the user to see a difference between the segments. The position of the start symbol is always the first coordinate of the first segment of the event track and the finish symbol is the last coordinate of the last segment. If the track only consists of one segment, the start and end position is the first and the last entry of the poly-line. In the case, there is no racetrack available for the event, the map view will not display any of these elements. An athlete consists of a single poly-line and a runner symbol and the points of the poly-line are the recorded GPS coordinates of the athlete. The current location of an athlete and their marker position is the last point of the poly-line. To distinguish athletes from each other, the poly-line and the runners' icon always have the same color, which is assigned by the controller. Additionally, an athlete also has a so called info window, which pops up over the athlete marker when clicked on. The info window contains information about the athlete and the user can display the athlete charts described in Section 4.5 using the links within the window. For the poly-lines of the athletes and the event track, it is always important for the controller to request the data according to the mapping method that the user selects. Furthermore, if a user changes the mapping type of the event track or athletes, the task of the controller is to automatically update the corresponding poly-line and the marker.

In addition to these elements, the map can place markers called points of interest (POIs). These POIs are placed along the racetrack and highlighted in certain positions of the event. Examples for POIs are milestones on the track, a difficult slope the athletes have to master or an obstacle that is along the track. POIs have the ability to trigger a notification popup to inform the user that an athlete is in proximity to the position on the track. This is only possible if the event provides a race track file. To place a POI on the track, three properties are necessary. To calculate the position of the marker on the track, the user must provide the distance from the POI to the start. The calculation uses the same algorithm as discussed in Section 4.4.3. Secondly, besides the marker symbol, the POI also shows a string that provides information of the point, such as water ditches, weavers, or a milestone represented with the string "10 KM". The third and last property is the notification distance. If the distance between an athlete and the POI is smaller than the notification distance, the controller creates a custom notification window to inform the user. To acquire the current distance of an athlete, the map controller uses the already stored distance from the statistics table.

The notification distance is important for many reasons. Firstly, if the user wants to be informed before an athlete reaches a certain POI, for example, if a moderator of an event wants to inform the people that an athlete is near an obstacle, or some visitors standing near a POI want to know when the athletes will be approaching. The second point is that GPS devices are sometimes unreliable and mistakes may occur such as the update rate of a GPS is not very high, or a device has connection problems and does not send data as frequently to the server as expected. It may happen that an athlete makes a big jump in distance between two update cycles and may never be within the update radius. In this case it is possible to increase the notification distance to still guarantee a notification alert. It is also important for every POI to keep track of the athletes that have already triggered a notification, as without it, every athlete that is within the range of the notification distance to a POI would trigger a notification every time the controller performs a check.

The last element placed on the map view is a placeholder where the operator of the application can insert their logo. Instead of placing the image on the website using an HTML tag, the Google Maps JavaScript API can insert custom elements inside the map. Therefore, images are part of the map rather than be part of the website. The custom map element also prevents the image from blocking the control elements of the map, which is not doable with a default HTML element. Since the map is inside a `<div>` container, an HTML element can either be in front of the map (such as the statistics table) or behind it.

Ordering Elements on the Map

By placing many elements on the map, it is important to define which elements are more important than others. Important elements should always be visible and in front of the other ones. The map has an already built-in ordering scheme to put different element groups (poly-lines, markers, info windows) in certain layers. These layers are also known as panes and are divided into the following order: (from lowest to highest)[9]:

- mapPane is the lowest pane and includes all images that build the map.
- The overlayLayer pane holds all the poly-lines and polygons. In this case it contains the poly-lines of the event track and the athletes.
- The markerLayer contains all markers, such as the start and finish symbols, the runners' symbols and the POI markers.
- overlayMouseTarget is the fourth layer and not used.
- The floatPane layer contains all the info windows and is above the other four layers.
- The highest layer is the pane, which includes all control elements of the map such as the zooming buttons or the buttons to change map terrain. The custom image which is part of the map uses these layers.

Within a layer, elements can also use the zIndex variable. The zIndex defines the display order of the elements. An element with a greater zIndex value is always in front of an element with a lower value. If two or more elements have the same zIndex value, the vertical position on the screen is important. Elements that are lower on the screen are in front of elements that are placed further up on the map. The three different types of used markers also have a predefined hierarchy. The runner icons have the lowest zIndex and are therefore always in the background. The second highest marker is the start and finish

[9]Google, *Google Map Custom Overlays*. `https://developers.google.com/maps/documentation/javascript/customoverlays#initialize` (visited on 20/03/2019).

symbol. They are in front of the runners because at the start of an event all athletes are at the starting line and cover the starting line symbol. POIs are always in front of all the other markers because they are important elements along the track and should always be visible.
Ordering the poly-lines is also important as the poly-line of the event track is always in the background and the order above the racetrack are the poly-lines of the athletes. The display order depends on the positions inside the statistics table. The first element in the table is up front and the last element is always in the background. Changing the order of the table will also change the order of the athletes.

Important APIs Calls

All elements that are visible on the map are solely created using the Google Maps JavaScript API [Sly 2014]. If not described differently, the option argument of the constructors are JavaScript objects which hold multiple parameters.

Creating the map:

- Code: var map = new google.maps.Map(domElement, options).
- map: The constructors returns the map object. To interact with the map, or to place other elements on the map, always needs the map object. The map controller is responsible for storing this object.
- domElement: This argument is a reference object of the target container, where the map places its content. Executing the JavaScript method document.getElementById creates a reference object of an HTML element.
- options: Is a JavaScript object that can hold multiple properties to change the behavior and look of the map. For example, the control elements, default zoom level or the terrain type. By choice, the web page uses the standard look and feeling of Google Maps.

Creating a poly-line:

- Code: var polyline = new google.maps.polyline(options) and polyline.setMap(map).
- poly-line: The construct returns an instance of a poly-line with the given options. The poly-line is either part of an athlete or the event track and stored in corresponding containers of the data storage.
- options: The most important property is path, which takes an array of coordinates from the event or the athlete track as value. zIndex defines the drawing order as described above. To change the style of the poly-line, the controller can change the stroke color (strokeColor), the opacity (strokeOpacity) and the stroke weight (strokeWeight). If the geodesic property is set to true, Google Maps automatically includes the elliptic property of the earth when drawing a poly-line.
- polyline.setMap(map): To draw the poly-line on the map, it must set a map where the map is the object of the created map of the tracking page. By executing the code polyline.setMap(null) the poly-line will be removed from the map.

Creating a Marker:

- Code: var marker = new google.maps.Marker(options) or var marker = new MarkerWithLabel(options). POIs use the standard constructor of the Google Maps JavaScript API and the marker for the athletes, the start and finish line uses the custom constructor MarkerWithLabel, which additionally allows for modification of the style of the marker text.
- marker: The construct returns an instance of a marker with the given options. This is either a POI, a marker of an athlete or the start and finish marker of the race track. The controller stores the marker in a corresponding container of the data storage.

- options: The location of a marker on the map is defined by the properties position and map, where map takes the map object as a value and position takes a single coordinate as a value. zIndex defines the drawing order inside markerLayer of the markers as described above. With the icon property, it is possible to change the standard marker icon and replace it with an image or a SVG. The runner icon, start and finish line uses a SVG as a custom symbol. Default markers use the text property to add a text above the marker. The custom MarkerWithLabel uses the property labelContent. The custom marker adds the feature to customize the text by assigning a CSS class to labelClass property.

Changing marker and poly-line properties: In the following examples, the obj variable represents a poly-line or marker created using the Google Maps JavaScript API.

- obj.setVisible(true|false) changes the visibility of a poly-line and the marker.
- obj.set("propertyName", value) changes any arbitrary property of a poly-line or marker. For example, executing the code obj.set("zIndex", 10) changes the value of zIndex to 10.
- polyline.setPath(path) replaces the current points of the poly-line with new coordinates stored in the variable path.
- marker.setPosition(position) changes the position of the marker to the coordinate stored in the variable position.

6.3.4.3 Statistic Table Controller

As explained in Section 4.3, the number of different statistic types are dynamic and can be changed. For that reason, it is not possible to create a static table and fill the rows. The controller dynamically creates the table using the Bootstrap table plugin, as described in Section 6.2.1.7

The code `$.("#targetID").bootstrapTable(options)` creates the Bootstrap table where the variable options is a JavaScript object containing all the needed properties to create the table such as column information, table data, styling options and binding functions to certain table events. The columns are the statistic types from the REST interface described in Section 5.7.4.1 and the data is all statistics of all athletes from the current event using the REST interface described in Section 5.7.4.4. The controller must add a predefined column format because it is not part of the received data.

The order of the table rows defines the drawing order of the athletes poly-lines on the map. If the user changes the order either by using drag and drop or column sorting, the controller must inform the map controller to change the zIndex of the poly-lines.

The statistics can be shown in a popup window which is a blank HTML page with a target container for the table. This popup window must include the same JavaScript files as used by the athlete tracking page. The controller creates the popup window using JavaScript. It is important to create the window using the code var w = window.open("popup.html", ""); to have a communication channel between the popup window and the athlete tracking page. The variable `w` is a reference of the popup window. All globally set variables and functions are accessible via this variable and provide the communication channel. After all JavaScript files are loaded in the popup window, the controller must copy the references of the map and web elements into the new window. All changes applied on these references inside the popup window will change the elements from the athletes tracking page. For example, if the poly-line changes the color on the reference object it will update the poly-line on the map. The next step after copying the variables is to execute the initialization function of the popup window which creates the table. The controller executes the initialization function via the window reference using the code w.init(). It is also possible to assign certain functions to events of the popup window. If the user closes the popup window, it fires the event onbeforeunload. By firing this event, the controller automatically restores the statistics table on the athlete page and makes it visible again.

The data of the update table changes during the time of a live event. The extension provides built-in functions that automatically request new data from the external interface. However, a problem using this features is that, if a user changes the row order by drag and drop or by clicking the column header, the built-in update function will reset the order. To prevent this, the controller must override the internal sorting implementation of the extension and delete the stored sorting options of the columns. The second step to avoiding an order reset is not to use the global update function which updates all rows at ones, but to let the controller update every row separately using the data provided by the REST interface described in Section 5.7.4.3.

6.3.4.4 Site Controller

The "Site" controller is the component that is responsible for all the other elements except the statistics table and the map. Certain elements of the model are dynamically generated and can be changed over time. These elements are athletes, mapping types and chart types. For this reason, it is not beneficial to hard code items dynamically into the view and change the it every time the model changes. The task of the site controller is to get the data from the model and build the view elements. This includes the following elements:

- The mapping types to map the event track (see Figure 6.12).
- Charts that are available for the event track.
- The mapping types for the athletes (see Figure 6.19b).
- Athlete Select Box (see Figure 6.20).

As shown in Figure 6.12 the drop-down menu of the navigation item includes the mapping types for the athlete track and the all charts that are available to display. The task of the controller is to generate the drop-down entries for the missing elements and link user interaction to the corresponding functions. The drop-down elements of the navigation menu are simple hyperlink tags (`<a href= ...>... </a>`) inside a specific `<div>` element. If the user clicks on one of the mapping types, it will inform the map controller to update the event track using the data of the selected mapping algorithm. By clicking on a chart type, the controller executes the functions provided by the chart controller. If the event does not provide a track, the user can upload one or multiple files via the upload page (see Section 6.3.6).

The controller must open the modal window if the user clicks on the menu item of the POI drop-down menu (see Figure 6.14). By adding a new POI, the controller executes the implemented interface of the map controller and passes the data. The same applies for modal which lists all POIs where the user can change the values or remove them, seen in Figure 6.16b. The controller has to either execute the update or remove functions of the map controller.

The athlete menu item also opens a modal, shown in Figure 6.19a. The controller must initialize the table with the athletes. Every athlete row also includes a drop-down list where the user can change the mapping type. The controller listens on the `change` event and passes the new mapping type to the map controller, which updates the athletes' poly-line.

On the right side of the navigation bar there is an athlete select form (see Figure 6.20). The controller is responsible for building the HTML fragments and adding them to the view. A drop-down list consists of multiple `<options>` tags which are inside the `<select>` tag. The task of the controller is to create the HTML element with the athletes as options.
Out of the box, the elements of a drop-down list can either be selected or deselected. A selected element in the drop-down list has a check mark right next to the name. The controller must modify the athlete selector and add additional select states for the three visibility options of an athlete, as discussed in Section 6.2.3.4. The controller intercepts the clicking event of the user and decides the next state. Switching between the three states uses the round-robin principle. The first state is "fully visible", the second state is "marker

only" and the last state is "not visible". The controller has to change the icon corresponding state and pass the visibility state to the map and stats container, which updates the athletes' visibility.

6.3.4.5 Persistence

Persistence on the athlete tracking page is an important part of improving the usability. The web application keeps track of the user's preference to restore them if needed. If an event lasts several hours, the browsers could close because of several reasons or the user may want to reload the page. By reloading or reopening the page all elements and settings will be gone and the user has to restore them manually. Making the web application more user-friendly could make some of the user's preferences be restored every time the user reloads the page. This includes the created POI and their settings, the visibility settings of the athletes, the mapping type of the athletes and the event track. All these performances are automatically stored in the user's browser and restored if available.

6.3.5 Start Page Controller

The start page of the web application only consists of three dynamic elements: two carousels and the event table, discussed in Sections 6.2.2.1 and 6.2.2.2.

The controller must initialize the two carousels by creating the images for the slide show. If there is no live or upcoming event, the controller hides the left carousel. The right carousel displays the four newest events. Depending on whether the event status is ended or live, the controller must add the matching text. By upcoming events, the controller uses a countdown timer as text which updates every second.

Lastly, the controller must link the event table with the REST interface described in Section 5.7.1.1. The controller has to extract the query parameters from the input fields and send the request if the user clicks the "Search" button. The received data is used to update the table.

6.3.6 Upload Page Controller

As mentioned before, the map can display the racetrack of an event. The racetrack can consist of one or multiple segments. Every segment of the racecourse is a GPX file stored in the file system of the server. The task of the upload page controller is to extract the files of form and upload them using the REST interface described in Section 5.7.5. To upload a GPX file the controller must include the event ID in the request, so the server can link the files to the selected event. If the user accesses the upload page via the link of the athlete tracking page, the controller can access the event data using the `window.opener`. If the user accesses the upload page using the static URL `/upload`, the upload page does not have a parent window. In this case the `window.opener` reference is `null` and the controller cannot extract the event ID. Therefore, the controller creates an element selector with all events.

The controller restricts the selectable files to files which use a valid GPX format. The controller makes it possible to remove one or multiple files from the form. By clicking the upload button, the controller creates an AJAX request including the following properties in `data` object:

- `file` is the input-stream of the file.
- `id` is the ID of the event.
- `currentFile` is the name of the file.
- `ukey` is a unique upload key with current time in milliseconds, returned form the JavaScript object `Data` as a value.

The `data` attribute of the AJAX request is explained in Section 6.1. If the user did not select an event from the drop-down list, the controller will not send the request to the server. For every file selected by the user, the controller must create a separate AJAX request. After all requests are successfully sent to the server, the UI is updated to inform the user. If the upload page is accessed via the athlete tracking page, the controller will automatically reload the racetrack using the `window.opener` object.

Chapter 7

Evaluation

This section gives an evaluation of the live tracking systems. The system was tested during a live event using a simulation.

7.1 Kosiak Löwe

The Kosiak Löwe[1] is a duathlon located in Feistritz and consists of a 12.8 kilometer mountain bike leg and a 1.8 kilometer running leg. The cycling leg is along marked roads and the running leg is along hiking trails.

7.1.1 Testing Setup

Used Tracking Devices

Meitrack MT90[2] was the used global positioning system (GPS) tracker for both athletes. The location update interval of both devices was set to 30 seconds. During the test, one GPS tracker kept losing the connection to the server and therefore sent less updates to the server. Due to this problem, statistical calculations and the produced data for the charts were not always correct.

Test Environment

- **Setup Type**: Desktop
- **Browser**: Google Chrome
- **Operating System**: Ubuntu 18.04 LTS
- **Display Resolution**: 1920x1080
- **Hardware Specification**: 8x2.2 GHz, 8GB RAM

Event Track

The racetrack consists of two GPS exchange format (GPX) files, one for the cycling leg and one for the running leg.

[1]Alpen-Adria-Universität Klagenfurt, *Kosiak Löwe*. `https://www.aau.at/usi/kosiak/` (visited on 20/03/2019).
[2]Meitrack, *MT90*. `http://www.meitrack.com/en/mt90/` (visited on 20/03/2019).

7.1.2 Testing Results

7.1.2.1 User Interface

The testing system did not have any problems displaying the athlete's live tracking page. All functions described in Section 6.2.3 worked as expected and did not cause any errors on the client's device or on the server. Notifications triggered by points of interest (POIs) worked as well. However, in some instances, the notification was created after passing a POI and not before. This problem happened due to the chosen update interval and the connection errors of one of the GPS trackers. Using a larger notification distance solves this problem regarding the update interval, however, the problem caused by the faulty GPS device could not be fixed. The chart creation also worked as expected and no errors occurred on the test environment.

7.1.2.2 Mapping

All mapping methods could be tested during the event. Mapping algorithms (see Sections 4.4.1 and 4.4.1) that maps athletes to the road could be used during the cycling leg up to ta certain point. The first 80 percent of the cycling leg are along marked roads which are also part of Google Maps. In this section, both mapping functions worked well, although some GPS records were not exactly beside the road. After that segment, the two mapping algorithms were not usable because both tried to map the track to roads which were not part of the racetrack.

The mapping algorithm described in Sections 4.4.3 and 4.4.4 could be used the entire time. As expected, the "Track Mapping Algorithm with Moving Radio-frequency Identification" (TMA-MRF) performed better than the "Map on Track" algorithm. The difference was noticeable after about one hour when the racetrack had more curves and GPS coordinates were further away from the racetrack. Due to the inaccuracy, the calculated distance using the athlete's coordinates was smaller than the actual covered distance. Therefore, the "Map on Track" algorithm mapped athletes behind their actual position, in contrast to TMA-MRF.

7.1.2.3 Statistic

The statistics' calculations produced no errors and worked during the whole event. All statistics produced are valid results except the estimated arrival. Since the event consists of a cycling and running leg, the average speed is different between these sections. However, the average speed is calculated over all athlete coordinates and because the cycling leg is much longer than the running leg, the calculated value does not get smaller. This leads to the wrong calculation of the arrival time.

7.1.3 Statistical Evaluation

- **Number of stored Recorded GPS Points**
  - Device 1: 169 (device with connection problems)
  - Device 2: 370
- **Number of Server Request**: ca. 2200
  - Device 1: ca. 1100
  - Device 2: ca. 1100
- **Data Downloaded**: ca. 35 Megabyte (MB) in 180 minutes
  - Device 1: ca. 20 MB
  - Device 2: ca. 14 MB

- **Average Server Response Time**:
  - Event and Athlete Date: 100 - 250 ms
  - Mapping Algorithm using Google API: 1500 - 5500 ms
  - Other Mapping Algorithm: 300 - 700 ms.

7.2 Simulation

Since it is not always possible to test the web application during a live event with GPS trackers, it is necessary to create a test environment. The test environment simulates a live event consisting of multiple athletes and can be enabled by setting the `debug` property to `true`, as described in Section 4.6. The GPS coordinates for the simulation are recorded with a smartphone using the Runtastic app discussed in Section 2.1.1 and can be downloaded as GPX files[3]. An advantage to using the Runtastic app is the that one has the possibility to check if the created charts and calculated statistics are correct as Runtatsic creates similar evaluations. The GPX file created by the Runtastic app is the racetrack of the simulation event and also the GPS coordinates of the athletes. By starting the server in the debug mode, the simulation automatically starts as well. The event consists of four athletes starting with a start interval of 90 seconds. It takes about 25 minutes until the last athlete reaches the finish line and, after an additional five minutes, the event starts again. Since the simulation always uses the same data, it is well suited for testing implementation. Although the racetrack and the athletes use the same coordinates, it is still possible to test the mapping algorithm such as Google Map to Road or Google Direction API. To test the mapping algorithm that uses the racetrack to map athletes, a new aligned race track is used.

Since the simulation always produces the same outcome, there are no problems with the GPS devices or the update rate. Therefore, all features of the live tracking page tested if the simulation event works with the used devices. For that reason, the tests of the different devices are rather a performance than a usability test of the different features.

7.2.1 Testing Setup

Table 7.1 shows the devices and their specifications used to test a live tracking page during the test event.

Device Number	Setup Type	Browser	Operating System	Display Resolution	Hardware Specification
1	Desktop	Firefox	Ubuntu 18.04 64 Bit	1920x1080	8x2.2 GHz 8 GB RAM
2	Tablet	Edge	Windows 10 64 Bit	1920x1200	4x1.3 GHz 4 GB RAM
3	Smartphone	Google Chrome	Android 9 64 Bit	1080x2160	8x1.8 GHz 4GB RAM
4	Smartphone	Firefox	Android 8 32 Bit	720x1280	4x2,2GHz 2GB RAM

Table 7.1: A list of used devices to test the simulation event.

[3]Stary, *Simulation GPX File*. `https://www.wikiloc.com/outdoor-trails/simulation-track-35326911` (visited on 20/03/2019).

7.2.2 Testing Results

Table 7.2 shows the result of the statistical evaluation of the live tracking page for every device. All statistics are recorded with the profiling software of the used browsers.

Device Nr.	**1**	**2**	**3**	**4**
Average page loading time in seconds	2.5	7	4.5	21
Average CPU usage while opening in percent	17	80	70	-
Average CPU Usage during live tracking update process in percent	<1.5	50	30	-
Average RAM usage in Megabyte	21	13	17	18

Table 7.2: The result of the statistical evaluations of the used devices for the simulation event.

7.2.3 Usability Test

Within this thesis a usability test was made to test the user interface in practice. The patricians of this evaluation received a prefabricated instruction and tested the application based on these tasks. The study took place in the facilities of the participants or of the author of this thesis. In some instances, it was also possible to perform these tasks online via remote desktop, since both the application and the questionnaire are accessible by using the internet. The organization of the study proceeded as followed: Every participant could use their own computer or smartphone to perform their tasks. If no such device was available, one was provided. The participants could also choose their preferred browser to access the web page. Every participant performed the task alone to avoid getting influenced by someone else, who might had already accomplished the tasks. During the test the participant was only allowed to talk to the moderator, if something was unclear. Furthermore, the moderator explained the purpose of the usability study and gave a brief introduction in the live tracking topic and discussed the tasks, if needed. For the usability test the already existing data of the external database was used. To test the athlete tracking page, the simulation event was chosen as discussed in Section 7.2. The tasks of the usability study are described in Appendix A.

After finishing the tasks, all participants received three questionnaires which can be divided into two groups. The first group of the questionnaire gathers information of the participants (see Section B.1) and the second one is used to get feedback on the usability and some ideas for improvement, as discussed in Sections B.2 and B.3. In contrast to Sections B.1 and B.2 which consists of self-created questions, Section B.3 uses the questions based on the System Usability Scale (SUS) [Bangor et al. 2008]. SUS consists of ten predefined questions which are rated using a Lickert scale. The result of the SUS questions is shown in Table B.13 and Figure 7.1 shows the results in a radar chart.

A total of 20 participants took part in the questionnaire. The average age was calculated to be 27.1 years. 35% (7) of the participants were women and 65% (13) were men. The average amount of computer experience of all participants lies at 13.7 years. The respondents use a computer 4.2 hours on average per day and about the same amount of time is used for surfing the internet using tablets, smartphones, or computers. The questionnaire also shows that 50% of the interviewees have used a live tracking application (primary Runtatsic) before and 80% would be open to using one in the future. This result shows that live tracking systems are not immensely popular at the moment but have potential. All in all, the participants liked the user interface and found it easy to use, as shown by the results of the SUS questionnaire.

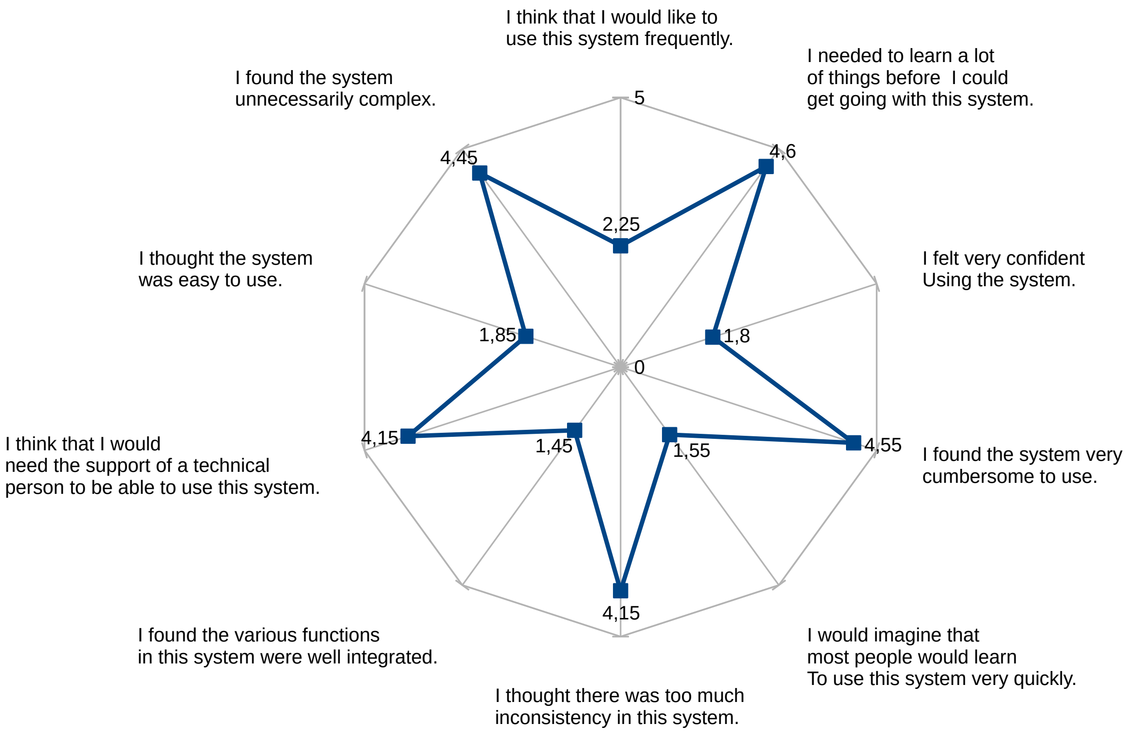


Figure 7.1: The result of the SUS questions visualized as a radar chart.

Chapter 8

Outlook

As discussed by Bonneau et al. [2017], the sports tracking market is a big industry nowadays and more and more companies provide new hardware devices or applications for smartphones to enable tracking. Since it is becoming easier to connect tracking devices to the internet, data can be transmitted in real time to a server [J. Wei 2014]. Furthermore, tracking devices are becoming smaller and more accurate. The live data can be used to increase the performance of athletes, the engagement with fans and the athletes' safety[1] and improve the health of humans [Palma et al. 2019; Anzaldo 2015]. Due to the live presentation of the data and better fan engagement, it becomes easier to attract new athletes and spectators. As a result of the increasing popularity it is also more likely to acquire new sponsors. Nowadays, live tracking software is used in many sports, such as football, soccer, swimming, winter sports or gymnastics [Bonneau et al. 2017]. Since the possible methods to track athletes is going to improve, it will also improve tracking applications and the experience for athletes, spectators, and moderators during a live event.

An important part of the web application is the user interface (UI). It is always possible to improve the user experience by enhancing the UI. This can be accomplished by making usability tests with multiple users and different devices. With the result of the usability tests it would be possible to improve the UI and change the behavior of some elements. Another option is to make a user survey to gather information about features that can be added to web applications or are important for spectators, athletes, and coaches. Another possibility is to make the application accessible to more users by increasing the number of supported languages.

During a live event the client continuously requests data from the server to update athletes. With an increasing number of athletes and global positioning system (GPS) coordinates, the amount of data sent to the client increases as well. The current implementation always sends all coordinates of an athlete to the client. The amount of the coordinates per request can be reduced by only sending the missing or new coordinates to the client. However, this requires a more complex logic on the server-side. What may occur in this instance is that the database does not receive all the data from the GPS device. Therefore, the athlete track stored in the database may be missing coordinates, even some that are several minutes old. For that reason, the server must keep track of the coordinates sent to the client, which might be difficult because of the RESTful principles (see Section 5.5). In addition to this, it would be difficult to only send new coordinates to the clients by using specific mapping algorithms because it is not possible to link the mapped coordinates to original ones.
Another option for decreasing the amount of data sent is to adjust the update interval of the client. The server can check the update rate of a GPS device and pass the information to the client. Finally, the coordinates can be compressed to decrease the data size, for example, by using the algorithm discussed

[1]Giorgio, *Internet of Things in Sports*. `https://www.deloitte.com/us/en/pages/consumer-business/articles/internet-of-things-sports-bringing-iot-to-sports-analytics.html` (visited on 20/03/2019).

in Section 4.4.2. However, every client must be able to decompress the data.

Another option to add more features to the web application is to create new charts, statistics, or mapping algorithms. At the moment, charts are created for the individual athletes, but it would also be useful to visualize all athletes in a single chart to compare them. It is also possible to add more statistics to provide more information about the athletes. To make charts and statistics more accurate, smoothing algorithms can be used to remove faulty GPS coordinates created from a GPS device [Liu et al. 2010; MCGRAW 2009; Jun et al. 2006]. However, it is hard to find mapping or smoothing algorithms for GPS coordinates which are suitable for this application.

Lastly, maintaining and refactoring the application is always important. Although the software design and implementation is well thought out, it is always possible to improve the code and make it more manageable and readable. In addition to that, it is essential to update the used libraries, which often improves the performance of the application.

Chapter 9

Conclusion

This thesis has presented the creation process of an athlete's live tracking software for outdoor events using global positioning system (GPS) trackers. The first chapter included GPS tracking and the goals of the application.

This thesis has presented the creation process of an athlete's live tracking software for outdoor events using global positioning system (GPS) trackers. The first chapter included GPS tracking and the goals of the application.

Chapter 2 introduced the currently used live tracking systems and discussed their implementations. It also presented other scientific work which uses tracking systems or athlete statistics based on the collected information about solutions which were acquired through research. Chapter 3 discussed the steps of gathering requirements and designing the software. The required engineering process consists of five steps and uses techniques such as user stories or prototypes of the user interface. It was crucial in this process to plan the software architecture of the application based on high-level design decisions, such as security concerns or a client-server model.

Chapter 4 explained the technologies used by the client and server and discussed the implemented server software structures as well as the design decisions. An important task of the application is to access an external database to receive the location data of the athletes. This was accomplished by implementing a data access interface. The data is used to accumulate a number of different real-time athlete statistics. It is also utilized to visualize the athlete's performance using charts. Major components of the software are the mapping algorithms which can align the athlete's track from the GPS devices to roads or on the racetrack.

The uniform interface called representational state transfer (REST) is covered in Chapter 5. It described the terminology, design guidelines and implementations on the interface. The rest of the interface uses the produced data from the server, for instance, statistics, athlete track, or chart data, and make them accessible for others via the internet.

An explanation of the software design of the client and the implemented user interface can be found in Chapter 6. It discussed each component of the model-view-controller pattern of the web application and the implementation of them. An important part of the client is the interaction between the REST interface and model using asynchronous JavaScript and XML (AJAX). In addition to that, it explained the user interface which was created using the cascading style sheets (CSS) framework called Bootstrap.

Chapter 7 discussed the testing results of the application. Two tests were used to evaluate the performance of the application during a live and a simulated event. The third test was a usability test where participants had to perform certain tasks and fill out a questionnaire afterwards, including questions based on a system usability scale.

Finally, Chapter 8 concluded the thesis by presenting current trends of live tracking software and ideas for further development.

Appendix A

Tasks of The Usability Study

Start Page Task:

1. Navigate through the slideshow.
2. Display all events that occurred later than 1.1.2016 in the Event-Table.
3. Search for the "Live Event" using the search field.
4. Open the “Dummy Live” event.

Athlete Tracking Tasks:

1. Move the map and center it again.
2. Change the mapping of the racetrack to "Google Street Mapping".
3. Display the altitude profile of the racetrack.
4. Display the "pace chart" of "Athlete 3".
5. Change the mapping of "Athlete" 2 to "Map On Track".
6. Change the visibility of "Athlete 1", so that only the runner’s icon is visible.
7. Change the visibility of "Athlete 2", so that neither runner’s icon nor the athlete’s track is visible.
8. Add a point of interest (POI) with the following specifications:
   - Name: 2KM
   - Distance: 2000
   - Notification Distance: 200
9. Change the name of the created POI to “2 kilometer”
10. Delete the changed POI.
11. Change the table order by sorting the athletes by distance in descending order.
    Devices with smaller screen must open the table using the pop-up button.
12. Move "Athlete 1" to the first position of the table using the drag and drop functionality.
13. Change the language of the web page.

Appendix B

Results of the Usability Study

B.1 Participants' Information

How old are you?

22	29	40	15	38	16	21	35	22	22	22	34	25	34	28	24	24	14	48	29

Table B.1: Result table 1/6 of the participants' information.

Are you male or female?

m	m	f	f	f	f	f	m	m	f	m	m	m	m	m	m	m	f	m	m

Table B.2: Result table 2/6 of the participants' information. The abbreviation "m" stand for male and "f" for female.

How many years of computer experience do you have?

12	20	1	1	25	5	15	26	13	10	10	14	24	23	12	17	13	4	10	20

Table B.3: Result table 3/6 of the participants' information.

How many hours a day do you use the computer?

6	8	0	5	2	2	8	3	10	0	2	4	1	9	6	5	4	2	1	6

Table B.4: Result table 4/6 of the participants' information.

How many hours a day do you surf the internet?

3	8	2	7	2	5	8	5	3	0	3	4	1	4	5	3	12	2	1	6

Table B.5: Result table 5/6 of the participants' information.

Have you ever used a Live Tracking software?

No	No	No	No	Yes	Yes	No	No	Yes	No
Yes	Yes	No	Yes	Yes	Yes	Yes	No	Yes	No

Table B.6: Result table 6/6 of the participants' information.

B.2 Additional Feedback

Was it clear, in every situation, what to do next?
If not, in which situation did you not know what to do next?

Yes	-
Yes	-
No	Due to the lack of computer experience I found I needed some time to get to know the user interface.
Yes	-
No	Change the athletes' visibility.
Yes	-
No	Change the athletes' visibility.
Yes	-
Yes	-
No	Change the athletes' visibility.
Yes	I was not sure what to do in some task and had to ask again.
Yes	-
Yes	-
Yes	-
Yes	-
Yes	-
No	Hide the column of the table.
Yes	-
Yes	-
Yes	-
Yes	-

Table B.7: Result table 1/6 of the additional feedback.

What do you particularly like about the system?

Clear user interface, changing of the map view, information about pace, altitude and the charts
-
Clear user interface.
The layout of the user interface.
Easily usable.
Visualization of the athletes.
The layout and easy to use.
Clear user interface.
The layout and good usability.
Well-structured and the visualization of the athletes.
Clear user interface.
Clear user interface and easy to learn/understand.
I like the idea of a live tracking application.
Well-structured user interface.
The visualization of the athletes on the map.
Very clear user interface and easy to use.
Simple and clear user interface.
Clear user interface.
The graphical presentation of web page.
Clear user interface.

Table B.8: Result table 2/6 of the additional feedback.

Are there some things that are missing?

-
-
The functionality of some symbols is not clear.
For a beginner too many technical terms.
-
-
-
-
Add POI the unit is missing.
A help window.
-
-
-
-
-
Adding a POI by clicking on the map.
-
-
-
-
-

Table B.9: Result table 3/6 of the additional feedback.

Is there something, you don't like? If yes, what?

-
-
For experienced users it would be surely no problem, but I needed some time to get to know the items of the user interface.
-
-
Changing the mapping of the athletes and the track.
Find the item where to change the athletes' visibility.
-
-
-
-
-
-
-
The visualization of the athletes (track) are overlapping too much.
-
-
-
-
-
Not possible to change the color of the athletes. Placing placement of the visibility selector.

Table B.10: Result table 4/6 of the additional feedback.

Have you ever been in a situation where a live tracking software would have been useful?

No
No
No
No
No
No
No
Yes
No
No
Yes
Yes
No
No
Yes
Yes
No
No
No
No

Table B.11: Result table 5/6 of the additional feedback.

Would you use such a software?

Yes
Yes
Yes
Yes
Yes
No
Yes
Yes
Yes
Yes
Yes
Yes
No
Yes
Yes
Yes
No
No
Yes
Yes

Table B.12: Result table 6/6 of the additional feedback.

B.3 System Usability Scale Results

	Strongly Agree	Agree	Neutral	Disagree	Strongly Disagree
I think that I would like to use this system frequently.	5	9	2	4	0
I found the system to be simple.	0	0	1	7	12
I thought the system was easy to use.	8	9	3	0	0
I think that I could use the system without the support of a technical person.	1	3	2	4	12
I found the various functions in the system were well integrated.	12	7	1	0	0
I thought there was a lot of consistency in the system.	1	0	3	5	11
I would imagine that most people would learn to use the system very quickly.	13	6	0	1	0
I found the system very intuitive.	1	0	0	3	16
I felt very confidence using the system.	11	6	3	0	0
I could use the system without having to learn anything new.	0	0	2	4	16

Table B.13: Result of the usability test based on the questions on SUS of the 20 participants.

Bibliography

Abd Mikjel H, Trygve Reenskaug [1979]. *The original MVC reports* (1979) (cited on page 79).

Ahtinen, Aino, Minna Isomursu, Ykä Huhtala, Jussi Kaasinen, Jukka Salminen, and Jonna Häkkilä [2008]. *Tracking Outdoor Sports - User Experience Perspective*. Ambient Intelligence. Edited by Emile Aarts, James L. Crowley, de Boris Ruyter, Heinz Gerhäuser, Alexander Pflaum, Janina Schmidt, and Reiner Wichert. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pages 192–209. doi:10.1007/978-3-540-89617-3_13 (cited on page 1).

Andrews, Keith [2019]. *Writing a Thesis: Guidelines for Writing a Master's Thesis in Computer Science*. Graz University of Technology, Austria. 24 Jan 2019. `http://ftp.iicm.edu/pub/keith/thesis/`.

Anzaldo, D. [2015]. *Wearable sports technology - Market landscape and compute SoC trends*. 2015 International SoC Design Conference (ISOCC). Oct 2015, pages 217–218. doi:10.1109/ISOCC.2015.7401796 (cited on page 123).

Bangor, Aaron, Philip T. Kortum, and James T. Miller [2008]. *An Empirical Evaluation of the System Usability Scale*. International Journal of Human-Computer Interaction 24.6 (2008), pages 574–594. doi:10.1080/10447310802205776 (cited on page 120).

Bartolucci, Francesco and Thomas Brendan Murphy [2015]. *A finite mixture latent trajectory model for modeling ultrarunners' behavior in a 24-hour race*. Journal of Quantitative Analysis in Sports 11.4 (Jan 2015). doi:10.1515/jqas-2014-0060. `https://doi.org/10.1515/jqas-2014-0060` (cited on page 13).

Berndsen, Jakim, Aonghus Lawlor, and Barry Smyth [2017]. *Running with Recommendation*. Volume 1953. CEUR Workshop Proceedings. 2017, pages 18–21 (cited on pages 13, 46).

Berners-Lee, T. [1994]. *Universal Resource Identifiers in WWW*. RFC 1630. Jun 1994. `https://tools.ietf.org/html/rfc1630` (cited on page 67).

Berners-Lee, T., R. Fielding, U.C. Irvine, and L. Masinter [1998]. *Uniform Resource Identifiers (URI): Generic Syntax*. RFC 2396. Aug 1998. `https://tools.ietf.org/html/rfc2396` (cited on page 67).

Berners-Lee, T., R. Fielding, and L. Masinter [2005]. *Uniform Resource Identifier (URI): Generic Syntax*. RFC 3986. Jan 2005. `https://tools.ietf.org/html/rfc3986` (cited on pages 67, 71).

Bonneau, Vincent, IDATE, Laurent Probst, Bertrand Pedersen, Jill Wenger, and PwC [2017]. *The Internet of Things: reshaping the sport industry. Digital Transformation Monitor*. European Commission, May 2017. `https://ec.europa.eu/growth/tools-databases/dem/monitor/content/internet-things-reshaping-sport-industry` (cited on page 123).

Bray, Tim [2017]. *The JavaScript Object Notation (JSON) Data Interchange Format*. RFC 1654. Dec 2017. `https://tools.ietf.org/html/rfc1654` (cited on page 31).

Carcangiu, Alessandro, Gianni Fenu, and Lucio Davide Spano [2016]. *A Design Pattern for Multimodal and Multidevice User Interfaces*. Proceedings of the 8th ACM SIGCHI Symposium on Engineering Interactive Computing Systems. EICS '16. Brussels, Belgium: ACM, 2016, pages 177–182. ISBN 978-1-4503-4322-0. doi:10.1145/2933242.2935876 (cited on page 79).

Daniel, F., F. Casati, P. Silveira, M. Verga, and M. Nalin [2011]. *Beyond Health Tracking: A Personal Health and Lifestyle Platform*. IEEE Internet Computing 15.4 (Jul 2011), pages 14–22. doi:10.1109/MIC.2011.53 (cited on page 13).

Darekar, Sameer, Atul Chikane, Rutujit Diwate, Amol Deshmukh, and Archana Shinde [2012]. *Tracking System using GPS and GSM Practical Approach*. May 2012 (cited on page 1).

Drawil, Nabil M., Haitham M. Amar, and Otman A. Basir [2013]. *GPS Localization Accuracy Classification: A Context-Based Approach*. IEEE Transactions on Intelligent Transportation Systems 14.1 (Mar 2013), pages 262–273. ISSN 1524-9050. doi:10.1109/TITS.2012.2213815 (cited on page 47).

Edgecomb, S.J. and K.I. Norton [2006]. *Comparison of global positioning and computer-based tracking systems for measuring player movement distance during Australian Football*. Journal of Science and Medicine in Sport 9.1-2 (May 2006), pages 25–32. doi:10.1016/j.jsams.2006.01.003 (cited on page 13).

Fette, I. and A. Melnikov [2017]. *The JavaScript Object Notation (JSON) Data Interchange Format*. RFC 8259. Dec 2017. `https://tools.ietf.org/html/rfc8259` (cited on page 65).

Fielding, Roy Thomas [2000]. *Architectural Styles and the Design of Network-based Software Architectures*. PhD Thesis. 2000. ISBN 0-599-87118-0 (cited on page 69).

Gosling, James, Bill Joy, Guy Steele, Gilad Bracha, Alex Buckley, and Daniel Smith [2019]. *The Java Language Specification*. 12th Edition. Oracle. 08 Feb 2019, pages 328–338. `https://docs.oracle.com/javase/specs/jls/se12/jls12.pdf` (cited on page 30).

Gregorius, Thierry and Geoffrey Blewitt [1998]. *The Effect of Weather Fronts on GPS Measurements*. GPS World 9 (09 Jan 1998) (cited on page 47).

IEEE Computer Society [2014]. *Guide to the Software Engineering Body of Knowledge*. 3rd Edition. Washington, DC: IEEE Computer Society Press, 2014. ISBN 978-0-769-55166-1 (cited on page 15).

J.Vickers, Andrew and Emily A. Vertosick [2016]. *An empirical study of race times in recreational endurance runners*. BMC Sports Science, Medicine and Rehabilitation 8.1 (26 Aug 2016), page 26. ISSN 2052-1847. doi:10.1186/s13102-016-0052-y (cited on page 13).

Jansen, Kai, Nils Ole Tippenhauer, and Christina Pöpper [2016]. *Multi-receiver GPS Spoofing Detection: Error Models and Realization*. Proceedings of the 32Nd Annual Conference on Computer Security Applications. ACSAC ’16. Los Angeles, California, USA: ACM, Dec 2016, pages 237–250. ISBN 978-1-4503-4771-6. doi:10.1145/2991079.2991092 (cited on page 47).

José Ignacio, Laura, Díaz-Casillas, and Carlos Á. Iglesias [2008]. *A Comparison Model for Agile Web Frameworks*. Proceedings of the 2008 Euro American Conference on Telematics and Information Systems. EATIS ’08. ACM, 2008, 14:1–14:8. ISBN 978-1-59593-988-3. doi:10.1145/1621087.1621101 (cited on page 25).

Jun, Jungwook, Randall Guensler, and Jennifer H. Ogle [2006]. *Smoothing Methods to Minimize Impact of Global Positioning System Random Error on Travel Distance, Speed, and Acceleration Profile Estimates*. Transportation Research Record 1972.1 (2006), pages 141–150. doi:10.1177/0361198106197200117 (cited on page 124).

Kacmarcik, Gary and Travis Leithead [2018]. *UI Events*. W3C Working Draft. https://www.w3.org/TR/uievents/. W3C, 08 Oct 2018 (cited on page 32).

Kazman, R., M. Klein, M. Barbacci, T. Longstaff, H. Lipson, and J. Carriere [1998]. *The architecture tradeoff analysis method*. Proceedings. Fourth IEEE International Conference on Engineering of Complex Computer Systems. Aug 1998, pages 68–78. doi:10.1109/ICECCS.1998.706657 (cited on page 26).

Lemoine, F. G., S. C. Lemoine, J. K. Factor, R.G. Trimmer, N. K., D. S. Chinn, C. M. Cox, S. M. Klosko, S. B. Luthcke, M. H. Torrence, Y. M. Wang, R. G. Williamson, E. C. Pavlis, R. H. Rapp, and T. R. Olson [1998]. *The Development of the Joint NASA GSFC and the NIMA Geopotential Model EGM96*. NASA/TP-1998-206861. Jul 1998. `http://cddis.nasa.gov/926/egm96/` (cited on page 42).

Liu, H., S. Nassar, and N. El-Sheimy [2010]. *Two-Filter Smoothing for Accurate INS/GPS Land-Vehicle Navigation in Urban Centers*. IEEE Transactions on Vehicular Technology 59.9 (Nov 2010), pages 4256–4267. ISSN 0018-9545. doi:10.1109/TVT.2010.2070850 (cited on page 124).

Martin, Robert C. [2017]. *Clean Architecture: A Craftsman's Guide to Software Structure and Design*. 1st Edition. Prentice Hall, 20 Sep 2017, pages 36–41. ISBN 0134494164 (cited on page 21).

Masse, Mark [2011]. *REST API Design Rulebook: Designing Consistent RESTful Web Service Interfaces*. 1st Edition. O'Reilly Media, 18 Oct 2011. ISBN 9781449310509 (cited on page 71).

MCGRAW, GARYA [2009]. *Generalized Divergence-Free Carrier Smoothing with Applications to Dual Frequency Differential GPS*. Navigation 56.2 (2009), pages 115–122. doi:10.1002/j.2161-4296.2009.tb01748.x (cited on page 124).

National Imagery and Mapping Agency [2000]. *Department of Defense World Geodetic System 1984: its definition and relationships with local geodetic systems*. Technical report TR8350.2. Version 3. National Imagery and Mapping Agency, 03 Jan 2000. `http://earth-info.nga.mil/GandG/publications/tr8350.2/tr8350_2.html` (cited on page 41).

Ocariza Jr., Frolin S., Karthik Pattabiraman, and Ali Mesbah [2015]. *Detecting Inconsistencies in JavaScript MVC Applications*. Proceedings of the 37th International Conference on Software Engineering - Volume 1. ICSE '15. Florence, Italy: IEEE Press, 2015, pages 325–335. ISBN 978-1-4799-1934-5 (cited on page 25).

Palma, Davide Di, Pompilio Cusano, Carmine Russo, and Antonio Ascione [2019]. *New Technologies in Sport through the Internet of Things Systems*. Research Journal of Humanities and Cultural Studies 5.1 (2019), pages 16–22. ISSN 2579-0528 (cited on page 123).

Patii, N. and B. Iyer [2017]. *Health monitoring and tracking system for soldiers using Internet of Things(IoT)*. 2017 International Conference on Computing, Communication and Automation (ICCCA). May 2017, pages 1347–1352. doi:10.1109/CCAA.2017.8230007 (cited on page 13).

Pautasso, Cesare and Erik Wilde [2010]. *RESTful Web Services: Principles, Patterns, Emerging Technologies*. Proceedings of the 19th International Conference on World Wide Web (Raleigh, North Carolina, USA). ACM Press, 30 Apr 2010, pages 1359–1360. doi:10.1145/1772690.1772929 (cited on page 65).

Pautasso, Cesare, Olaf Zimmermann, and Frank Leymann [2008]. *RESTful Web Services vs. 'Big' Web Services: Making the Right Architectural Decision*. Proceedings of the 17th International Conference on World Wide Web (Beijing, China). ACM Press, Apr 2008, pages 805–814. doi:10.1145/1367497.1367606 (cited on page 65).

Peguero, Ksenia, Nan Zhang, and Xiuzhen Cheng [2018]. *An Empirical Study of the Framework Impact on the Security of JavaScript Web Applications*. Companion Proceedings of the The Web Conference 2018. Republic and Canton of Geneva, Switzerland: International World Wide Web Conferences Steering Committee, 2018, pages 753–758. ISBN 978-1-4503-5640-4. doi:10.1145/3184558.3188736 (cited on page 26).

Pers, J., G. Vuckovic, S. Kovacic, and B. Dezman [2001]. *A low-cost real-time tracker of live sport events*. ISPA 2001. Proceedings of the 2nd International Symposium on Image and Signal Processing and Analysis. In conjunction with 23rd International Conference on Information Technology Interfaces (IEEE Cat. No.01EX480). Jun 2001, pages 362–365. doi:10.1109/ISPA.2001.938656 (cited on page 13).

Pham, Hoang Dat, Micheal Drieberg, and Chi Cuong Nguyen [2013]. *Development of vehicle tracking system using GPS and GSM modem*. Dec 2013, pages 89–94. ISBN 978-1-4799-0285-9. doi:10.1109/ICOS.2013.6735054 (cited on page 1).

Phillips, A. and M. Davis [2009]. *Tags for Identifying Languages*. RFC 5646. Oct 2009. https://tools.ietf.org/html/rfc5646 (cited on pages 70, 106).

Raj, C., C. Jain, and W. Arif [2017]. *HEMAN: Health monitoring and nous: An IoT based e-health care system for remote telemedicine*. 2017 International Conference on Wireless Communications, Signal Processing and Networking (WiSPNET). Mar 2017, pages 2115–2119. doi:10.1109/WiSPNET.2017.8300134 (cited on page 13).

Ranacher, Peter, Richard Brunauer, Wolfgang Trutschnig, Stefan Van der Spek, and Siegfried Reich [2015]. *Why GPS makes distances bigger than they are*. International Journal of Geographical Information Science 30.2 (01 Jul 2015), pages 316–333. doi:10.1080/13658816.2015.1086924 (cited on page 47).

Riegel, Peter S. [1981]. *Athletic Records and Human Endurance: A time-vs.-distance equation describing world-record performances may be used to compare the relative endurance capabilities of various groups of people*. American Scientist 69.3 (1981), pages 285–290. ISSN 00030996 (cited on page 13).

Riva, Claudio and Markku Laitkorpi [2009]. *Designing Web-Based Mobile Services with REST*. Service-Oriented Computing - ICSOC 2007 Workshops. Edited by Elisabetta Di Nitto and MateiRipeanu. Springer Berlin Heidelberg, 2009, pages 439–450. ISBN 978-3-540-93851-4. doi:10.1007/978-3-540-93851-4_42 (cited on page 69).

Rivoal, Florian, Tab Atkins Jr., and Elika Etemad [2019]. *CSS Snapshot 2018*. Technical report. World Wide Web Consortium W3C, 22 Jan 2019. https://www.w3.org/TR/2019/NOTE-css-2018-20190122/ (cited on page 22).

Roman, Steven [2002]. *Access Database Design & Programming*. 3rd Edition. O'Reilly Media, 2002. ISBN 9780596002732 (cited on page 38).

Seppelin, Thomas O. [1974]. *The Department of Defense World Geodetic System 1972*. Technical report. Defense Mapping Agency, May 1974. https://apps.dtic.mil/dtic/tr/fulltext/u2/a110165.pdf (cited on page 41).

Sly, John Michael [2014]. *Getting Around with Google Maps - A Programmer's Guide to the Google Maps API*. 1st Edition. Createspace Independent Pub, Dec 2014. ISBN 978-1-493-75338-3 (cited on page 112).

Smith, Michael [2017]. *HTML: The Markup Language (an HTML language reference)*. Technical report. World Wide Web Consortium W3C, 14 Dec 2017. https://www.w3.org/TR/2017/REC-html52-20171214/ (cited on page 22).

Smyth, Barry and Pádraig Cunningham [2017]. *A Novel Recommender System for Helping Marathoners to Achieve a New Personal-Best*. Proceedings of the Eleventh ACM Conference on Recommender Systems (Como, Italy). RecSys '17. ACM, 2017, pages 116–120. ISBN 978-1-4503-4652-8. doi:10.1145/3109859.3109874 (cited on pages 13, 46).

Sohan, S. M., Frank Maurer, Craig Anslow, and Martin Robillard [2017]. *A study of the effectiveness of usage examples in REST API documentation*. 2017 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC). Oct 2017, pages 53–61. doi:10.1109/VLHCC.2017.8103450 (cited on page 71).

Sollins, K. and R. Fielding [1994]. *Functional Requirements for Uniform Resource Names*. RFC 1737. Dec 1994. https://tools.ietf.org/html/rfc1737 (cited on page 67).

Stephen, Rod [2015]. *Beginning Software Engineering*. 1st Edition. New York: John Wiley & Sons, 2015. ISBN 978-1-118-96914-4 (cited on page 15).

Syromiatnikov, A. and D. Weyns [2014]. *A Journey through the Land of Model-View-Design Patterns*. 2014 IEEE/IFIP Conference on Software Architecture. Apr 2014, pages 21–30. doi:10.1109/WICSA.2014.13 (cited on page 79).

Vincenty, Thaddeus [1975]. *Direct and Inverse Solutions of Geodesics on the Ellipsoid With Application of Nested Equations*. Survey Review 23.176 (1975), pages 88–93. doi:10.1179/sre.1975.23.176.88 (cited on pages 41, 43).

Vosloo, Iwan and Derrick G. Kourie [2008]. *Server-centric Web Frameworks: An Overview*. ACM Comput. Surv. 40.2 (May 2008), 4:1–4:33. ISSN 0360-0300. doi:10.1145/1348246.1348247 (cited on page 25).

Wang, Pengfei and Aleksandar Kuzmanovic [2018]. *"What's the score?": A first look at sports live data feed services*. IEEE INFOCOM 2018 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS). IEEE, Apr 2018. doi:10.1109/infcomw.2018.8406822 (cited on page 13).

Wei, J. [2014]. *How Wearables Intersect with the Cloud and the Internet of Things : Considerations for the developers of wearables.* IEEE Consumer Electronics Magazine 3.3 (Jul 2014), pages 53–56. ISSN 2162-2248. doi:10.1109/MCE.2014.2317895 (cited on page 123).

Wei, X., P. Lucey, S. Morgan, and S. Sridharan [2016]. *Forecasting the Next Shot Location in Tennis Using Fine-Grained Spatiotemporal Tracking Data*. IEEE Transactions on Knowledge and Data Engineering 28.11 (Nov 2016), pages 2988–2997. doi:10.1109/TKDE.2016.2594787 (cited on page 13).

Wesley, Addison [2012]. *Software Architecture in Practice*. 3rd Edition. Addison Wesley, 25 Sep 2012. ISBN 9780321815736 (cited on page 79).

WGS Implementation Committee [1967]. *Department of Defense World Geodetic System 1966 Part II, Parameters and graphics for practical application of WGS 66*. Technical report. Department of Defense Joint Services, 1967 (cited on pages 41–42).

Wöllik, Helmut and Andreas Müller [2013]. *Sports timing with moving RFID-finish line*. 9th International Symposium on Computer Sciences in Sport (Istanbul). Dec 2013 (cited on pages 53, 56).