TECHNISCHE UNIVERSITÄT WIEN

DIPLOMARBEIT

OpenTravelMap

zur Erlangung des akademischen Grades

Diplom-Ingenieur/in

im Rahmen des Studiums

Geodäsie und Geoinformation

eingereicht von

Andreea Plocon
Matrikelnummer 01127962

ausgeführt am Department für Geodäsie und Geoinformation
der Fakultät für Mathematik und Geoinformation der Technischen Universität Wien

Betreuung
Betreuer: Univ.Prof. Mag.rer.nat. Dr.rer.nat. Georg Gartner

Wien,

Andreea Plocon	Georg Gartner

Abstract

The overwhelming touristic data collections currently available to any user is a problem of great concern nowadays. Due to the large amounts of information the user has to assess, the process of decision making became more and more time consuming and difficult to complete [20]. In an attempt to overcome this issue, systems that effectively assist users by providing access to relevant information should be developed. With this purpose, there are several applications in the field of tourism that offer detailed travel guides, descriptions of points of interest and even allow users to create their own travel routes. The present thesis focuses on the development of a mobile location based service that assists users in creating and searching for travel routes, the *OpenTravelMap* platform. In contrast to other existing platforms, the *OpenTravelMap* platform not only provides tools that facilitate the process of route creation and route search, but also integrates features that aim to enhance these functionalities. These features refer to the connection with other platforms in order to access external resources and a context aware recommender system. The outcome of this thesis, the *OpenTravelMap* platform, aims to be a worthwhile contribution to the field of tourism in view of the benefits supplied to its users.

Contents

List of Figures

List of Tables

List of Acronyms

LBS	Location Based Service
POI	Point Of Interest
REST	Representational State Transfer
SDK	Software Development Kit
UI	User Interface

Chapter 1

Introduction

Nowadays almost everyone has access to large amounts of information which supports the process of decision making. More and more platforms tend to filter the provided content, in order to offer only relevant information for certain kind of situations. Therefore, there is no such platform that addresses all the problems in the world but there are several ones developed to offer support for each kind of problem in particular.

In the field of tourism there are many platforms which provide detailed travel guides, descriptions of points of interest and even allow users to create their own travel routes. The purpose of this thesis is to develop the *OpenTravelMap* platform, a platform that enables and promotes the involvement in volunteered geographic information while overcoming the gaps existing in similar platforms. The *OpenTravelMap* platform offers assistance in the process of creating and searching for travel routes, while providing the following features:

- components for route creation and search in one platform
- context aware recommendations
- integration with other platforms in order to access existing resources

The assimilation of all these features aims to overcome the constrains existing in similar platforms. Furthermore, the support provided by the platform in creating in sharing content among users encourages the involvement in volunteered geographic information. In contrast to other existing platforms, the *OpenTravelMap* not only supports both the possibility to create and search for routes, but also aims to enhance both functionalities. The *Create* component provides a way to create routes and to describe the added content. Furthermore, the *Search* component of the *OpenTravelMap* platform allows sharing information between users and also adapts the retrieved content through context awareness.

The target audience consists of tourists which can benefit from the functionalities provided by the platform. The entire content is built by users in the *Create* component and shared between users through the *Search* component.

Various platforms targeting tourists are continuously being developed. The list below describes similar work that was released on Google Play [8]. A detailed comparison between these platforms and the *OpenTravelMap* is depicted in Chapter 2.
Platforms sharing the concept of own map creation:

- *Travel Map Maker*: create personal travel map and share it via email.
- *Map Maker*: create personal map by adding markers on a base map. Several sharing methods available.

Platforms sharing the concept of location/route search:

- *Sygic Travel Maps Offline & Trip Planner*: explore numerous popular places and plan detailed trips.
- *bergfex Tours & GPS Tracking Running Hiking Bike*: search for tours and track sport activities.

As mentioned before, the outcome of this thesis is the *OpenTravelMap* platform which consists of two components. The main purpose of the *Create* component is to provide methods which not only allow users to generate content, but also support them in describing it, by offering the possibility to integrate related information from other sources. The goal of the *Search* component is on the one hand to enable content sharing between users and on the other hand to enhance the search, if requested, by adjusting the retrieved content through context awareness.

The *Create* component of the *OpenTravelMap* platform aims to provide support for generating travel routes. To do so, the following definitions are used:

- A location represents a point collected from GPS data
- A direction represents the way traveled between two locations
- A route consists of locations connected by directions

These definitions describe the general terms: location, direction and route that are used to model the real world. For example, a location can represent a touristic attraction, a restaurant, a hotel, an outstanding position along a hiking route or a spectacular beach. A direction is always the connection between two locations such as streets, paths along parks or in the mountains. Finally, a route is a collection of locations and can therefore combine all sorts of activities.

Since each location is collected from GPS data, there could be several points referring to the same location. Therefore, the first research question addresses the point generalization issue [24]: "Which methods can be applied to perform the point generalization?". For the purpose of this thesis, names will be used to handle this issue. A location characterized by a name will then represent the collection of all points having the same name.

The second research question refers to: "Which methods and techniques can enhance the results of recommender systems?". As pointed out in [28] and [18], the context information is a valuable parameter that considerably improves the outcome of the search process. Therefore, the purpose of the *Search* component is not only to retrieve routes, but also to consider context information in the recommendation process. The user's search input together with the relevant context information will be handled inside a recommender system.

The last research question explores the scope of the *OpenTravelMap* platform: "How to develop a platform that integrates functionalities for route creation, route search and also delivers context aware recommendations?". By combining all these features and also providing access to external resources, offering information such as photos, weather forecasts or nearby places, the developed *OpenTravelMap* platform aims to overcome the gaps existing in the above mentioned similar platforms.

The present thesis is structured as follows: first, similar platforms that were already released on Google Play will be described in Chapter 2. Directly after, theoretical aspects related to the mentioned research questions will be presented in Chapter 3. Chapter 4 offers an overview of the architecture of the *OpenTravelMap* platform, followed by a detailed description of the methods used during the development. The entire specification of the *OpenTravelMap* platform is outlined in Chapter 5, where the *Create* and the *Search* component are presented based on a step by step scenario. Next, the achievements of this thesis are underlined in Chapter 6. Afterwards, improvements and future work are addressed in Chapter 7. Finally, a brief summary followed by a conclusion are depicted in Chapter 8.

Chapter 2

Related work

The aim of this section is to provide a brief description of some platforms developed for the field of tourism and released on Google Play [8] by highlighting the similarities and differences to the *OpenTravelMap* platform. While some of the platforms focus on providing travel guides for specific countries, regions or cities, others support the user in the navigation process and only a couple of them offer assistance for both. Moreover, there are only a few platforms that have a worldwide coverage and share the concepts of map creation and route search.

The concept of own map creation is the core feature of the ***Travel Map Maker*** platform [16]. Within this platform, the user is encouraged to add markers at desired locations. These markers are assigned to a map and the user has the possibility to create several such maps. As illustrated in Figure 2.1, the user can personalize the map by adding different markers. Furthermore, the created map can be managed by the user and sharing existing maps with friends can be done via email. According to the platform's documentation, it is also possible to add titles and descriptions to the markers. On the other hand, it is not permitted to assign pictures to existing markers. Another feature allows the user to measure distances on the map.

The major similarities between *Travel Map Maker* and *OpenTravelMap* are the ideas of creating and sharing location based content. The essential differences between the platforms are represented by the search functionality and the recommender system provided by the *OpenTravelMap* platform. Nonetheless, also the common ideas are implemented slightly different. More precisely, the concept of content sharing is supported within the *OpenTravelMap* platform through the *Search* component, whereas the *Travel Map Maker* platform allows map sharing via email. Apart from that, the *OpenTravelMap* platform provides enhanced location editing tools that allow the user to categorize the location and add a picture to it. Another fundamental difference between the platforms is the capacity of the *Travel Map Maker* platform to handle multiple maps, a feature that is not present within the *OpenTravelMap* platform where the user has the possibility to create only one map/route at a time. At first,

the creation of one map/route at a time might be seen as a disadvantage, since the user has to share or delete the created map in order to be able to start a new one. On the other hand, this prevents the existence of incomplete maps that a user might create and forget about.

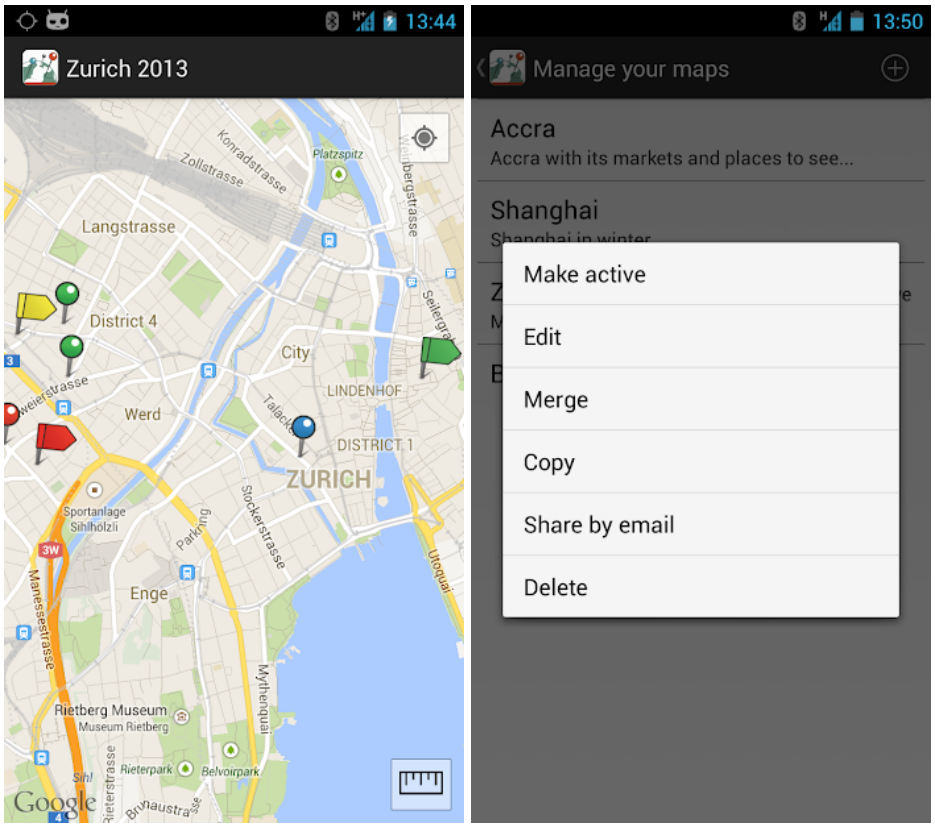

Figure 2.1: *Travel Map Maker [16]*

Another platform that shares the concept of map creation is ***Map Maker*** [9]. The platform allows the user to create a map by adding markers on a base map. A marker is characterized by several attributes, such as: title, description, color, date, an icon and pictures. Furthermore, the user has the possibility to change the position of the marker by moving it on the map. Displayed on the left side of Figure 2.2 is a map created within the *Map Maker* platform, which contains various markers. As illustrated on the right side of Figure 2.2, the user can organize the markers in folders. An extra chargeable feature allows saving the markers in the cloud and sharing them with other users. On the other hand, the import/export of markers from/to files is offered free of charge.

The idea of adding markers on desired places on the map and providing additional information to them is both part of the *OpenTravelMap* as well as of the *Map Maker* platform. However, the significant difference between the platforms is the purpose of the map creation process. While the *OpenTravelMap* platform supports the creation of travel routes that will be

further shared among users, the *Map Maker* platform focuses on the single user experience and provides multiple tools that assist the user in personalizing a base map. Apart from that, the *Map Maker* platform allows the user to have multiple folders, each of which containing different markers. By contrast, within the *OpenTravelMap* the created markers are grouped into routes and the user is allowed to create only one route at a time. Another remarkable difference between the platforms is the fact that the *OpenTravelMap* platform provides tools to create, search and recommend locations, whereas the *Map Maker* platform focuses only on the creation process.

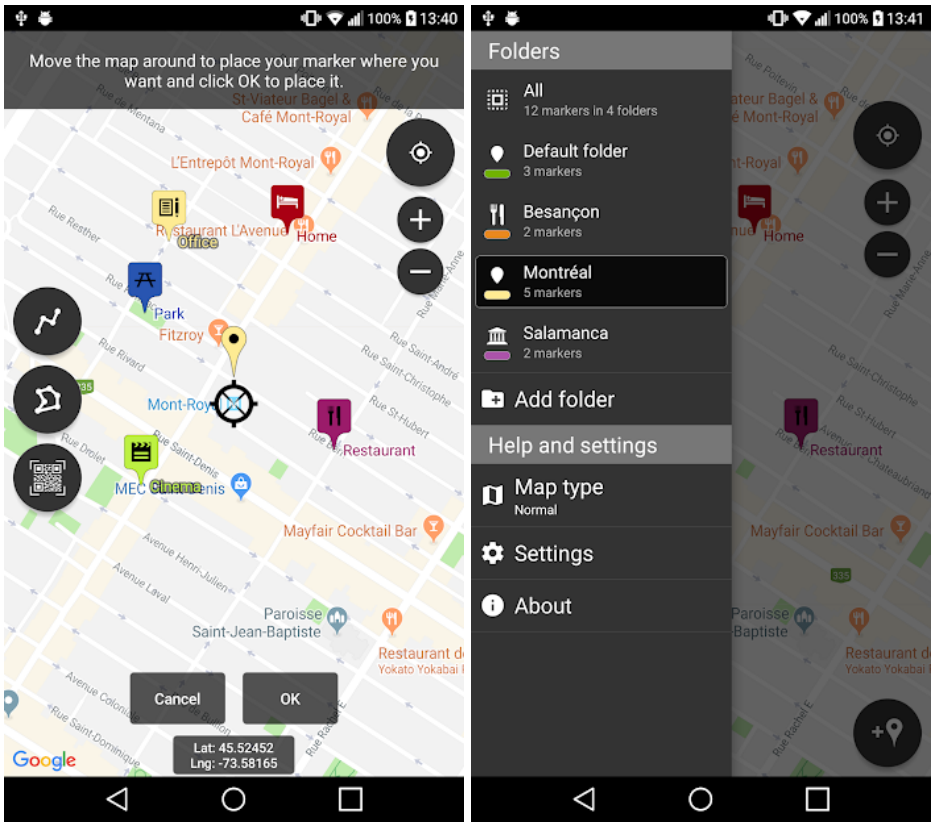

Figure 2.2: *Map Maker [9]*

Searching for places, receiving context-related information and offering navigation support are subjects of crucial importance in the field of tourism. A powerful platform that assists travelers along their journeys is ***Sygic Travel Maps Offline & Trip Planner*** [15]. According to its documentation, this platform is the ultimate travel guide. As indicated on the left side of Figure 2.3, the platform hosts information such as pictures, descriptions and even videos for a large number of points of interest. Furthermore, the platform also comprises a trip planer, illustrated on the right side of Figure 2.3, which assists the user in planing detailed journeys. Apart from that, the *Sygic Travel Maps Offline & Trip Planner* platform offers tools that allow

the user to search for places, tours and activities.

The main aspect that relates the *OpenTravelMap* platform with the *Sygic Travel Maps Offline & Trip Planner* platform is the concept of searching for touristic information. Both platforms aim to assist the user in the process of searching for the desired trip. The *OpenTravelMap* platform relies on the data created by users and therefore offers travel information at different spatial coverages around the world. By contrast, the *Sygic Travel Maps Offline & Trip Planner* platform possesses a database full of touristic information and is capable of offering many more details about a certain place than the *OpenTravelMap* platform. Nonetheless, the places shown within the *OpenTravelMap* platform contain user reviews which powerfully influence the user's decisions in the process of trip planing. A significant difference between the platforms is the fact that within the *OpenTravelMap* platform the user can choose one of the given travel routes, while the *Sygic Travel Maps Offline & Trip Planner* platform gives the user the possibility to plan a personal route. Even so, the *OpenTravelMap* also offers context related recommendations in order to help the user to choose the most appropriate route.

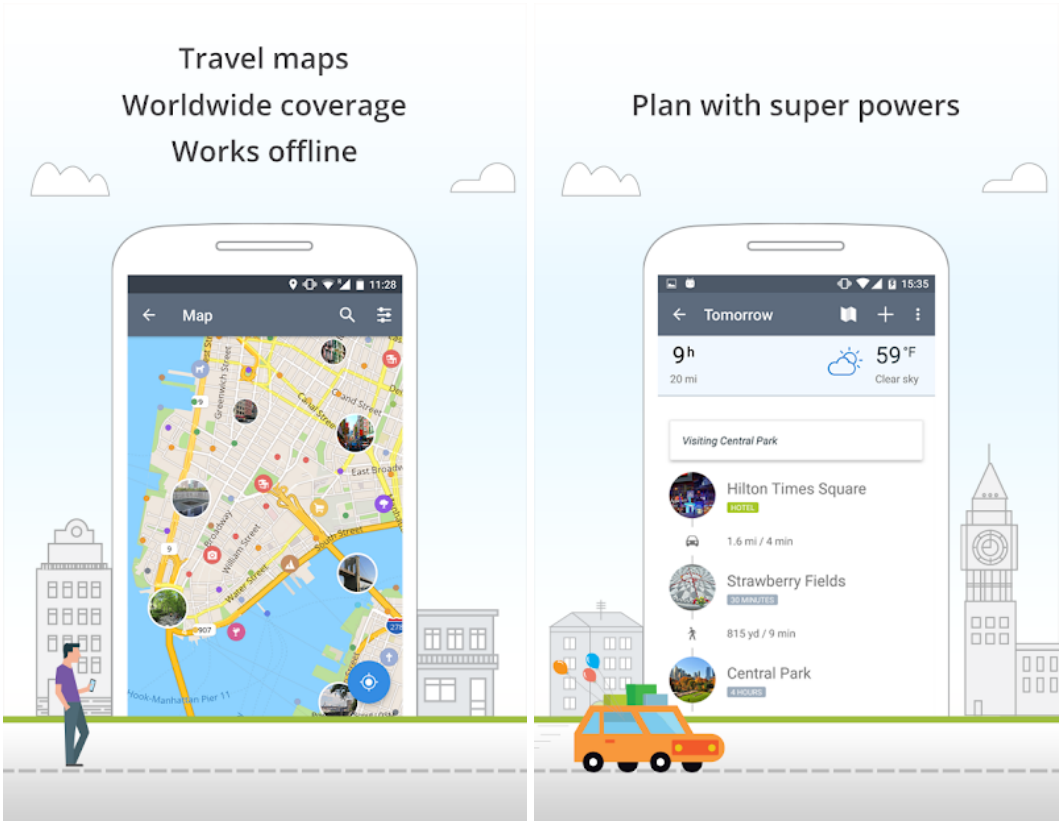

Figure 2.3: *Sygic Travel Maps Offline & Trip Planner [15]*

The concept of searching for routes is also grasped within the ***bergfex Tours & GPS Tracking***

Running Hiking Bike platform [2]. The platform focuses on outdoor activities and offers numerous tours for hiking, skiing, cycling and further sports. Each tour is characterized by its length, height, duration and several other useful information. Shown on the left side of Figure 2.4 are different tours from which the user can choose from, whereas the right side of the figure illustrates a detailed view of a particular tour. Furthermore, the user can decide to use the platform as an activity tracker, in which case the platform will offer statistical data related to the tracked activity.

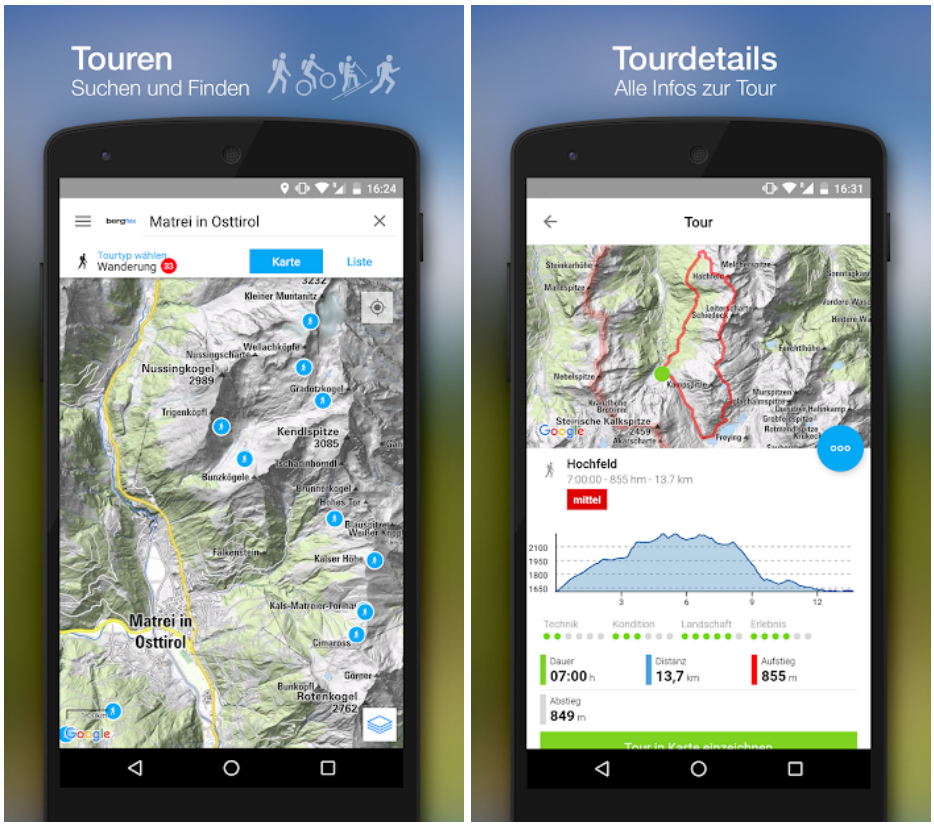

Figure 2.4: *bergfex Tours & GPS Tracking Running Hiking Bike [2]*

Both the *OpenTravelMap* platform and the *bergfex Tours & GPS Tracking Running Hiking Bike* platform aim to support the user in the process of searching for routes/tours. While the *OpenTravelMap* platform covers any kind of locations and routes, the *bergfex Tours & GPS Tracking Running Hiking Bike* focuses specifically on tours related to outdoor activities. Consequently, the platform provides detailed information for the tour itself. By contrast, the *OpenTravelMap* platform treats a route as a collection of locations and offers detailed information only for the individual locations. Beside the basic search tools, powerful features such as the recommender and the tracking systems are included, which significantly extend the functionality of both platforms.

Chapter 3

Theoretical background

The aim of this thesis is to develop the *OpenTravelMap* platform, a mobile location based service that will support users in the process of creating and searching for travel routes. In order to achieve this purpose, several theoretical aspects must be considered. This chapter focuses on the main research questions that influenced the development of the platform. In the beginning, a general description of mobile location based services and their design will be made. Afterwards, the problem of point data generalization will be addressed, by highlighting techniques that can be applied to handle overlapping locations created within the *OpenTravelMap* platform. Last but not least, insights on recommender systems and the importance of context aware recommendations will frame the theoretical concepts of the recommender system integrated in the *OpenTravelMap* platform.

3.1 Mobile location based services

A location based service (LBS) is a system that makes use of the geographical position of the user and provides services based on that given position. The fundamental difference between a service and a LBS is the fact that the LBS is a location dependent service. In other words, the geographic location is the key parameter in a location based service and it is always mandatory. Even though location based services are developed for a variety of domains, such as public safety, health, entertainment, tourism and many more, the main goal of all these services is to support users by exploiting the information about their position, in order to deliver a context related outcome that will help users in the process of decision making.

Since time is a crucial element in the process of decision making, the overall efficiency of a location based service is highly determined by its capability to rapidly deliver correct results and by its accessibility. For the purpose of providing fast access to information, mobile location based services were developed. These services are specially designed to run on mobile devices and facilitate the access to location related information anytime and anywhere.

The purpose of a location based service is to obtain the geographic location of the device it is running on and to provide information related to this location. As highlighted in [26], there are two major categories of LBS:

- triggered location based services (push services)
- user-requested location based services (pull services)

The first LBS category comprises services that are triggered automatically by an event. Usually, this kind of location based services are used when time has an extremely high importance. For example, in the field of emergency services the location of the user could be requested automatically at the arrival of an emergency call. The second category of LBS includes services that are requested by the user. For instance, a location based service in the field of tourism could provide the information about the nearest restaurant that serves a specific food when the user requests it.

As already mentioned, a location based service makes use of the geographic location of the user. In case of a mobile LBS, the service needs obtain the position of the user's mobile device, which is usually requested from the device's location provider. In order to protect the user's privacy, the device's location provider first asks for the user's permission and only then when the user agrees, the device's location is retrieved.

A challenging problem for mobile location based services is the way in which the user interface is designed. Due to the fact that mobile devices have small screens, the content shown to the user must be adapted, such that it not only fits the available space but it does not overload the view, while still providing the key information that the user requires in the process of decision making. This usability issue is also discussed in [31], in the context of the *CRUMPET* project, whose purpose is to create user-friendly mobile services personalized for tourism. The proposed solution, that aims to improve the user's experience, is the integration of context awareness into mobile location based services. Furthermore, the most important context parameter is considered to be the user's location.

3.1.1 UI design

Designing effective and user-friendly mobile location based services is a demanding task. The small screen and the limited interaction capabilities between the user and the mobile device significantly increase the complexity of the process of designing such user interfaces. The goal is to create a user interface that efficiently fills the screen of the mobile device, is intuitive and easy to use, shows relevant data but does not overload the view and most important, it supports the user in obtaining the wanted information.

Most of the mobile location based services tend to use a map-based user interface, which allows the visualization of context related information on top of a map. Even though a map helps the users to visualize their spatial environment, studies such as [23] focus on assessing the efficiency of the map-based approach by comparing it with a text-based design among various applications. The study reports that the choice of a user interface is highly dependent on the following aspects: the context in which the location based service is used and the type of information it is requested to deliver, as well as the personal preferences and experiences of the targeted user. As advised in [23], most probably a hybrid version combining the map-based approach with the text-based user interface would be the most suitable design for mobile location based services.

Several studies, such as [32] and [27] focus on the challenge of designing good map-based user interfaces for mobile location based services. The effective visualization of information and the intuitive and helpful interactions between the user and the mobile device are the key elements that lead to an efficient and user-friendly design. According to these studies, the density of information displayed on the screen must be adapted to meet the required level of detail. Therefore, different visualization techniques such as exaggeration, elimination, typification or outline simplification must be applied. Furthermore, the usability of a map-based design is also influenced by the variety of interactions between the user and the map. An intuitive interface should follow the guidelines describing the map interactions such as zooming, panning and item selection in order to ease the user's experience.

A comprehensive overview of the design principles that should be considered within map-based services is given in [29]. This book provides a detailed description of the main building parts of the map design. These include the user-map interactions, the map elements, the choice of colors and text components, the different types of symbols and thematic representations, as well as animation and sound concepts.

3.2 Point data generalization

A fundamental requirement of map-based applications is to provide readable maps that are easy to use and understand, while they contain all the relevant information that the user needs. This, however, is a challenging demand, especially for map-based mobile services which have to display their map on small screens. In order to improve the map's readability, the principle of map generalization should be applied. As mentioned in [24], the aim of this principle is to simplify the map's content while maintaining the information that the user requires. The process of simplifying the map's content consists on performing a series of graphic generalization operations that will adjust the content and at the same time will

emphasize the important information. A couple of such graphic generalization operations are displacements, deformations, simplifications, aggregations and deletions. Besides the graphic or geometric generalization, also the semantic generalization is part of the map generalization [22]. Assuming that the information is categorized, the semantic generalization determines which categories of information are relevant and should be displayed at certain levels of details.

In the field of tourism all mobile location based services deal with point data that originates from different sources. The user's location is usually requested from the device's location provider, while for example points of interest are fetched from other sources. Nevertheless, all these data will be displayed at some point in time on the map. Due to the limited space that the mobile devices possess and with the aim of ensuring the readability of the map, point data generalization techniques must be applied. A detailed description and analysis of state of the art algorithms used for point data generalization are given in [33].

Another behavior related to the map generalization process is caused by a user-map interaction, namely zooming. [19] and [21] present different generalization techniques that can be applied during zooming. While the user is zooming in or out, the level of detail of the map's content changes. Through this behavior the readability of the map is preserved at each zoom level. Consequently, the content displayed on top of the map (for example markers created by the user) must be adapted in a similar way.

As highlighted in [25], general knowledge is enough to handle the basic data generalization. On the other hand, customized restrictions and specifications related to the generalization of particular data are covered by thematic knowledge. The mentioned study introduces a generalization method based on thematic knowledge which follows the rule: *IF condition THEN operation*. Such a generalization method would be very well suited for services developed within the field of tourism, since these exploit large amounts of thematic data.

3.3 Recommender systems

In view of the fact that the amount of information available to any user has increased tremendously over the past decades, the process of decision making became more and more time consuming and difficult to complete. The large amounts of information, the overwhelming possibilities and the time spent to evaluate them are the major issues that the user faces when trying to make a decision [20]. Fortunately, recommender systems were introduced to overcome these problems. A recommender systems aims to ease the process of decision making by providing relevant recommendations that incorporate precisely the information

needed.

Depending on the techniques used to build the recommendations, recommender systems are categorized as follows [17]:

- Content based recommender systems
  Recommendations are based on previous user's preferences.
- Collaborative recommender systems
  Recommendations are based on choices made by other similar users.
- Hybrid recommender systems
  A combination of content based and collaborative techniques.

Among others, indicated in [17] are the limitations of these recommender systems together with potential improvements. Concerning the content based recommender systems, a common issue is the fact that recommendations are based on the user's profile and therefore quite similar over time. Furthermore, the items are characterized by features that are investigated during the recommendation process, but different items having the same features are considered identical. Another problem is that the system relies heavily on the user's profile, which is why it will take some time till new users will get proper recommendations. Exactly the same issue also affects the collaborative recommender systems. In addition, collaborative methods present limitations in recommending new added items, since these must first be rated by a significant number of users. Evidently, the number of users is a crucial factor that influences the operating of a collaborative recommender system.

In order to overcome the limitations of the content based and collaborative recommender systems, hybrid recommender systems were developed. Such a recommender system combines the content based and collaborative techniques with the aim of exploiting their benefits while reducing their constraints. Further improvements in the field of recommender systems are related to advanced user profiles and items representation, more complex recommendation techniques, integration of contextual information and flexible recommendations modeled through a SQL-like language [17].

The crucial role of the user's profile within recommender systems is emphasized also in [20]. Through profile analysis the system evaluates the user's preferences and is able to recommend relevant content. There are a variety of ways to build a user profile, starting with simple approaches, such as a collection of preferred features, to more complex models, such as a vector that stores for each feature its corresponding rating given by the user. Another important aspect highlighted in [20] is the fact that the user profile needs to be continuously updated in order to provide accurate recommendations. As soon as the user's preferences

change, a revision of the user's profile is required. According to the mentioned study, the user's profile should be updated based on feedback information, which is gained explicitly and/or implicitly. The explicit approach directly requests the user's feedback, whereas the implicit feedback is obtained by inspecting the user's behavior while the system is used.

In contrast to other domains, recommender systems developed in the field of tourism are normally exploited by users that are continuously moving. Therefore, supplying the same recommendations under different circumstances might not meet the user's needs [20]. For the purpose of providing more accurate recommendations, it is suggested that recommender systems should also take context information into account. This way, the recommendations could be adapted according to the user's context.

3.3.1 Context information

The term *context*, as defined in [28], represents any information that is used to describe the state of an entity. Systems that make use of context information to provide services to users are known to be context aware. The main objective of recommender systems is to help users in the process of decision making by offering relevant recommendations. In view of the fact that the process of decision making is significantly influenced by context information [28], this powerful parameter became part of the recommender systems and that is how context aware recommender systems appeared.

The development of context aware recommender systems is a challenging task, since most of the time the proper type of context information to be considered is unknown. Several contextual factors such as: location, weather, time of day, season, budget, companion, mood and many more are potential candidates that could improve the functionality of a recommender system. While integrating just a few contextual parameters might not be enough, taking too many parameters into account can easily increase the complexity of the system and might not be necessary. As highlighted in [18], part of the development of context aware recommender systems is the evaluation of the relationship between user's preferences and contextual parameters. This way, only essential context information will be integrated into recommender systems.

In order to deliver personalized recommendations, a context aware recommender system should be able to assess relevant user information such as likes/dislikes, interests/disinterest, wishes and fears. Such information is used to define a user and to build the user's profile. Ideally, the recommender system is able to create the user profile based on the monitored user activity. In this way likes and often pursued choices are interpreted as preferences and will be further recommended, while dislikes will be discarded from future recommenda-

tions. Furthermore, the personalized future recommendations offered to an user might also be influenced by the choices of other similar users, or other users that made similar choices.

A key feature of a context aware recommender system is represented by its ability to update the recommendations as soon as the context changes [18]. While there are some contextual parameters, such as weather or time of day, that can be monitored to rapidly detect modifications, there are lots of other parameters, such as the user's companion for example, whose changes can not be identified automatically. Therefore, in order to continuously provide accurate recommendations, the system will request the user's contribution to establish the current context information.

As expected, recommendations provided by a context aware recommender system incorporate both the user's preferences as well as context information. However, the user's acceptance of such recommendations is not completely granted. Suggested in [18] is the retrieval of explained recommendations. According to the mentioned study, the explanations are a powerful element that increase the user's confidence in the system and positively influence the user's acceptance of the provided recommendations.

Appropriate context information enhances the outcome of systems exploiting it. As reported in [28], the field of tourism strongly benefits from the introduction of context aware recommender systems, which considerably improve the user's experience. Apart from that, [30] emphasizes the usability improvements of mobile map applications that make use of context information.

Chapter 4

Development methods

The present chapter consists of two sections that depict the methods used during the development of the *OpenTravelMap* platform. The first section aims to provide an overview of the platform's architecture by briefly describing its main components: the client and the server side, respectively. Furthermore, the external apis used to access information from other sources will be discussed. The second section of the current chapter focuses on the setup of the *OpenTravelMap* platform, mainly on the development insights of the *Create* and *Search* component, respectively.

4.1 Architecture

The architecture overview of the *OpenTravelMap* platform is illustrated in Figure 4.1. As

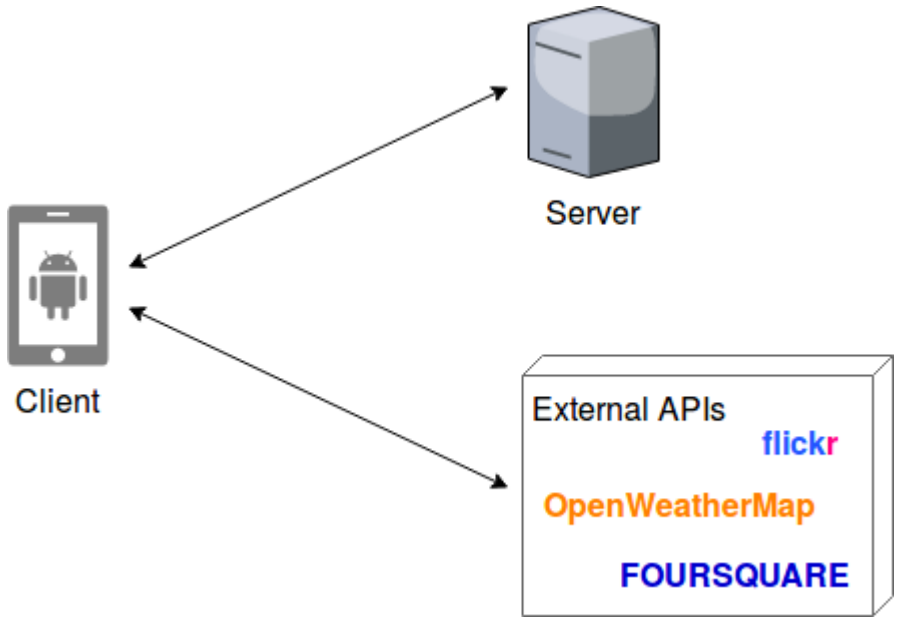

Figure 4.1: *Architecture overview*

can be seen in the figure, the main components are the server and the client side of the

platform. Moreover, the server-client platform is extended by the usage of external apis, which facilitate the access to further information sources. A more detailed representation of the platform's architecture is given in Figure 4.2. As shown in the figure, both the server and the client side of the platform have a similar structure which consists of two major components and a database. The *Create* component is responsible with the creation of travel routes and their storage on the databases. By querying the stored data, the *Search* component aims to present the results to all interested users. The external apis are used only by the client side of the platform. Depending on the information provided, either the *Create* or the *Search* component will request it from the external api.

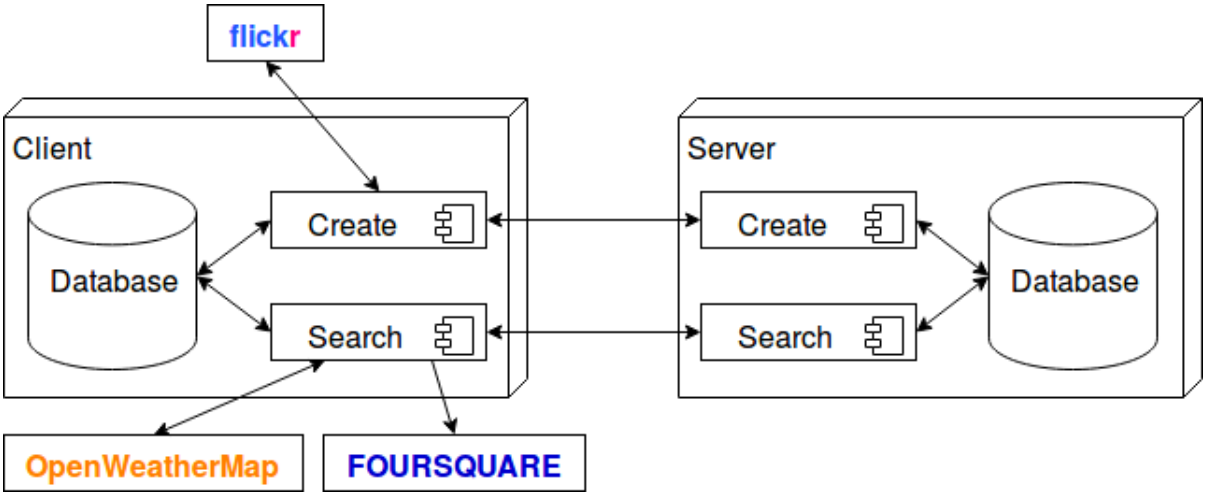

Figure 4.2: *Detailed architecture*

4.1.1 Client

The client side of the *OpenTravelMap* platform was designed to run on Android mobile devices, as indicated in Figure 4.1. The goal of the *OpenTravelMap* platform is to assist users in creating and searching for travel routes. Therefore, it is an essential requirement that the platform runs on mobile devices, in order to provide the users immediate access to the features of the platform. The Android application was implemented in Kotlin by following the guidelines described in [1]. As shown in Figure 4.2, the client side of the platform is comprised of a database, the *Create* component and the *Search* component.

The *Create* component as well as the *Search* component use the Maps SDK for Android provided by *Mapbox* [10] for displaying the map and the locations. In addition, the *Search* component also uses direction and geocoding services from *Mapbox* in order to draw directions between locations on the map and to geocode custom locations entered by users when searching for routes. For the purpose of this thesis, the support from *Mapbox* was gained via

a free account which offers a limited number of users and requests per month.

In the context of the *OpenTravelMap* platform, data is created inside the *Create* component. The process of creation stores the data on the client side of the platform. The user controls the trigger of the save process which will send the data to the server, store it there and finally clear the client storage. In this use case, the client database is a crucial element, since it enables access to the created data until this data is send to and successfully stored on the server. In other words, the user has the opportunity to create the data, view and/or edit it anytime in the future and save it exactly when desired. The client's database is also used by the *Search* component to store weather forecasts requested from the *OpenWeatherMap* platform. This weather forecasts are stored in order to significantly reduce the number of requests send to the external api. Technically, for the client database a SQLite database is used, as suggested by the Android Developers community.

As illustrated in Figure 4.2, the *Create* and *Search* components are two separate parts of the *OpenTravelMap* platform. The *Create* component handles the route creation, while the *Search* component manages the search and recommend processes. The components do not communicate with each other directly, but through the server side of the platform. Even though the components are completely independent of each other, the *Search* component still relies on the fact that there are enough routes stored on the server, routes which originate from the create process that takes place inside the *Create* component.

As mentioned before, the *Create* component offers support in creating routes. In an attempt to assist the user at the creation process, the *Create* component has to establish a connection with the client's database to store and check for a created route. The user has the possibility to create one route at once and each route consists of at least one location. Further user support is provided within the *Create* component through its communication with the server side of the platform to: provide name suggestions when adding a new location and save the current route with its corresponding locations. Apart from that, the user can also benefit from the connection between the *Create* component and the *Flickr* platform, in order to add publicly available pictures to the created locations.

The *Search* component of the client side of the *OpenTravelMap* platform is responsible with assisting the user to search for routes. Furthermore, it also contains a recommender system that enhances the search results by considering both the user's desires as well as relevant context information. The *Search* component only accesses routes stored into the server database. Within the search process, the *Search* component requests routes from the server which are located around a certain location entered by the user. In addition, the component also com-

municates with the *OpenWeatherMap* platform to provide weather information, which is used within the recommend process. In order to avoid sending a large number of requests to the *OpenWeatherMap* api, the *Search* component stores the received weather forecasts into the client database. Therefore, there is also a connection established between the client database and the *Search* component, as evident from Figure 4.2. Another remarkable feature of the component is represented by the links to the *Foursquare* platform, each of which redirecting the user to a collection of places to eat/drink around a given location.

4.1.2 Server

The main purpose of the server side of the *OpenTravelMap* platform is to store and query routes. The routes are created on the client side of the platform within the client's *Create* component, send to the server's *Create* component and further stored into the server database. In addition, the client's *Search* component can make requests to the server's *Search* component to query and retrieve routes from the server database. Due to the fact that the information stored on the server database contains point data, which is characterized by longitude and latitude coordinates, and also accounting on the fact that the server needs to be able to perform spatial queries, such as retrieving all data located within a given distance to certain coordinates, the database used on the server side is PostgreSQL with the PostGIS extension. For the purpose of this thesis the server database was set up using the Docker container provided by [3].

The server side of the *OpenTravelMap* platform was implemented in Java as a Spring Boot Application, by following the guidelines presented in [14]. The client and the server side of the platform communicate with each other via a REST api, whereby the client components send requests to their related server components. The server's *Create* component accepts the following requests on the mentioned endpoints:

- POST requests on "/saveRoute"
  Through such requests, routes created on the client side of the platform are sent to the server and stored into the server's database.

- POST requests on "/savePoi"
  A POI is characterized by its name, latitude and longitude coordinates. This kind of requests are automatically triggered by the client side of the platform each time a location is added that does not exist in the server's database. After performing this request, the location is saved as a POI into the server's database.

- GET requests on "/getNearbyPoiNames"
  This provides access to the POIs stored into the server's database. Such requests are

made by the client in order to obtain name suggestions for a new added location. If the response of this request is empty then the location is considered new and will be sent and saved on the server through a POST request on "/savePoi".

By contrast, the server's *Search* component only accepts GET requests on the endpoint "/getRoutes". This request takes as parameters the latitude and longitude coordinates of a location and a radius in meters. With these parameters the server performs a spatial query and returns all routes found within the given radius around the given coordinates.

4.1.3 External APIs

As evident from Figure 4.1, the *OpenTravelMap* also uses information provided from external sources. The user triggers the information request, which is sent from the client side of the platform to different external APIs. The aim of this section is to briefly describe the external APIs used by the *OpenTravelMap* platform.

The major components of the client side of the *OpenTravelMap* platform request information from the following external sources: ***Flickr***, ***OpenWeatherMap*** and ***Foursquare***. As illustrated in Figure 4.2, the client's *Create* component only uses the *Flickr* API, while the *Search* component makes use of two external information providers: the *OpenWeatherMap* and the *Foursquare* platform, respectively.

In order to fetch pictures located around an edited location, the *Create* component uses the ***Flickr*** API. These pictures are first shown to the user, then the user decides if one of those pictures gets assigned to the location being edited. Since the usage of the *Flickr* API is allowed only with an API key, a non-commercial key was generated for the purpose of this thesis.

The documentation of the *Flickr* API provides a detailed description of its available methods. For the purpose of getting pictures around a given location, the *flickr.photos.search* method is used. The request parameters as well as the return values of this method are described in [6]. A request sent by the *Create* component of the *OpenTravelMap* platform to the *Flickr* API looks as follows:

```
https://api.flickr.com/services/rest/?method=flickr.photos.search
    &api_key=API_KEY&text=LOCATION_NAME
    &lat=LOCATION_LATITUDE&lon=LOCATION_LONGITUDE
    &radius=DIST_TO_NEARBY_LOCATIONS_IN_KM&per_page=NUMBER_OF_IMAGES
    &sort=date-posted-desc&privacy_filter=1&content_type=1
    &format=json&nojsoncallback=1
```

The placeholders LOCATION_NAME, LOCATION_LATITUDE and LOCATION_LONGITUDE represent the name and coordinates of the location for which pictures are requested. The placeholders DIST_TO_NEARBY_LOCATIONS_IN_KM and NUMBER_OF_IMAGES are configuration parameters that can set within the *OpenTravelMap* platform. Within this thesis the value of the DIST_TO_NEARBY_LOCATIONS is set to 0.25 km while the NUMBER_OF_IMAGES is 5. Therefore, the request listed above will retrieve a maximum of 5 pictures related to the given location and taken within a radius of 0.25 km around the location's coordinates. The pictures will be sorted in descending order by the posted date such that the user always sees the newest ones. Furthermore, the parameter privacy_filter is set to ensure that only public pictures are retrieved.

The response of the request sent to the *flickr.photos.search* method contains information describing the pictures. According to the documentation of the *Flickr* API [5], the pictures can be accessed through a further request having the following format:

```
https://farmFARM.staticflickr.com/SERVER/ID_SECRET.jpg
```

The placeholders inside this request: FARM, SERVER, ID and SECRET are set with values taken from the response of the first request send to the *Flickr* API.

Another external API used to improve the functionality of the *OpenTravelMap* platform is the ***OpenWeatherMap***. The *OpenWeatherMap* API is called within the *Search* component, in order to provide weather forecasts that will be used as context information inside the recommender system of the platform. As usual, the API can be accessed through a key, which was obtained after creating a free account on the *OpenWeatherMap* platform.

For the purpose of obtaining weather information for the recommender system, the 5 day / 3 hour forecast endpoint of the *OpenWeatherMap* API is used [12]. The request sent from the client's *Search* component to the *OpenWeatherMap* API is:

```
http://api.openweathermap.org/data/2.5/forecast
    ?lat=LATITUDE&lon=LONGITUDE&units=metric&appid=API_KEY
```

The placeholders LATITUDE and LONGITUDE are replaced with the coordinates of the centroid point of all the locations displayed within the *Search* component. The coordinates of the centroid point are the average latitude and longitude values of these locations. The response of the request is parsed and only the 3 hour forecast of the current day is further processed. Within the current day forecast rain events are searched. The information passed to the recommender system is whether or not it will rain during the current day. Furthermore, also a weather icon is retrieved for visualization purposes. The list of weather icons provided by the *OpenWeatherMap* API is shown in Figure 4.3.
The weather icon is chosen as follows: if there is at least one rain event during the current

Day icon	Night icon	Description
01d.png 	01n.png 	clear sky
02d.png 	02n.png 	few clouds
03d.png	03n.png	scattered clouds
04d.png 	04n.png 	broken clouds
09d.png 	09n.png 	shower rain
10d.png 	10n.png 	rain
11d.png 	11n.png 	thunderstorm
13d.png 	13n.png 	snow
50d.png 	50n.png 	mist

Figure 4.3: *Weather icons provided by OpenWeatherMap [13]*

day, then the icon of the first rain event entry from the weather forecast is retrieved. If there is no rain event during the current day, then the weather icon is either the icon correspondent to the weather forecast from 12:00 o'clock (if the request was made before this time of the day) or the icon correspondent to the closest weather forecast of the current day. For example: considering that the current day is sunny and the request to the *OpenWeatherMap* API was sent at 10:00 o'clock, then the chosen weather icon will be the one of the forecast from 12:00 o'clock. However, if the request was sent at 16:00 o'clock, then the icon correspondent to the forecast from 18:00 o'clock will be chosen. This approach was implemented due to the fact that the *OpenWeatherMap* API provides 5 day / 3 hour forecasts starting from the current day at the time when the request is made. In the context of the *OpenTravelMap* platform these limitations of the *OpenWeatherMap* API are not a problem, since it makes no sense to check past weather in order to recommend future travel routes.

As mentioned before, the weather information passed to the recommender system is related to the centroid of all locations displayed within the *Search* component and relevant

for the current day. The weather request is triggered by the user, but there is no need to make more than one request per day for a certain centroid. Therefore, the first valid weather response together with the corresponding centroid and date will be stored into the client's database and retrieved at each further trigger. This way, a weather request will be sent to the *OpenWeatherMap* API only then when there is no entry into the client's database for the given centroid and date. Before the request is sent to the *OpenWeatherMap* API, the weather information stored in the client's database is deleted, since now it is considered to be out of date. All this explains the existing connection between the client's database and the *Search* component which is illustrated in Figure 4.2.

Shown in Figure 4.2 is also a connection between the *Search* component and the ***Foursquare*** platform. This connection aims to provide direct access to context filtered information regarding places to eat and drink. In the field of tourism providing context related information is a powerful feature. Therefore, the direct access to relevant information hosted by the *Foursquare* platform improves considerably the functionality of the *Search* component. The link to the *Foursquare* platform can be accessed within any location displayed in the *Search* component and it looks like follows:

```
https://foursquare.com/explore?mode=url
    &q=Food&ll=LOCATION_LATITUDE,LOCATION_LONGITUDE
```

The `LOCATION_LATITUDE` and `LOCATION_LONGITUDE` represent the coordinates of the location currently viewed by the user. Through this link, the user is redirected to the *Foursquare* platform, directly to a list of places to eat and drink, all located around the location viewed within the *OpenTravelMap* platform.

4.2 OpenTravelMap setup

The aim of this section is to provide a detailed description of the setup of the *OpenTravelMap* platform. This comprises insights on the management of critical scenarios, the depiction of the integrated features and last but not least, the configuration used to support the user. In the beginning, the setup of the *Create* component will be presented, followed by a description of the *Search* component's configuration.

4.2.1 Create component

On account of the fact that the purpose of the *Create* component is to assist users in creating and saving travel routes, which are formed from a collection of locations, the position of the current user is an essential element for the well functioning of the location based service provided by the platform. However, directly accessing the position of the user's device is not allowed, since it violates the user's privacy. In order to overcome this issue, the *OpenTrav-*

elMap platform asks for permission to access the device's location, as illustrated in Figure 4.4.

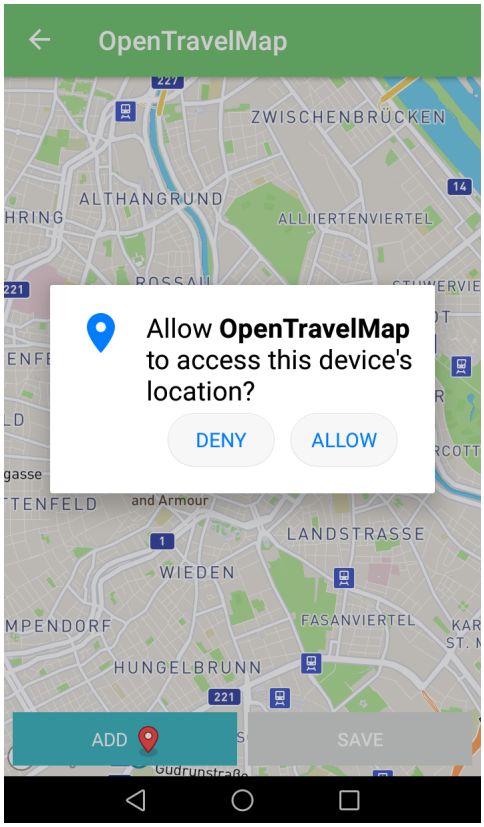

Figure 4.4: *Permission to access device's location*

The user will be requested to grant access on the device's location only once, during the first usage of the *OpenTravelMap* platform. As soon as the user accepted, the device's location is visible both to the *Create* and the *Search* component, respectively. While the access to the user's current position is of crucial importance within the *Create* component, the functionality of the *Search* component does not rely on this information.

Within the *Create* component, the user's position is used when the user adds a new location. This new added location will have the latitude and longitude coordinates of the user's device. By assuming that the device's location provider is disabled, it is clear that the current user's position is unknown to the *Create* component and therefore, the user will not be able to add any new location. In order to support the user, the *Create* component checks whether or not the device's location provider is enabled. As indicated in Figure 4.5, the user will be notified as soon as the *Create* component detects that the device's location provider is disabled.

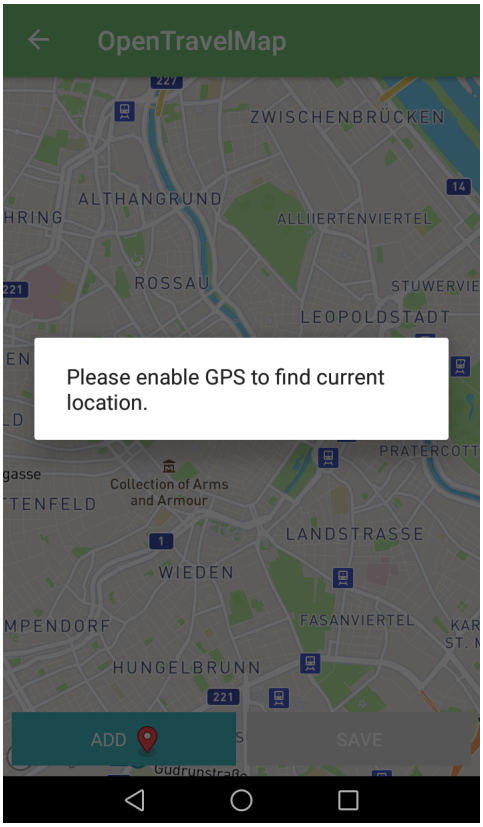

Figure 4.5: *GPS not enabled warning*

Even though adding a new location might seem to be an straight forward action, during this action there is a major issue worth to be mentioned. The user triggers the creation of a new location and the *Create* component performs this action as soon as the user's current position is known. As explained before, the coordinates of the user's position will assigned to the new added location. At this point, the user is able to add multiple locations having the same or slightly different latitude and longitude coordinates. By doing so, the user might intentionally or unintentionally harm the data generated within the *OpenTravelMap* platform. Therefore, a method to overcome this crucial issue has to be implemented.

In order to avoid the above mentioned issue, a strict rule has been defined within the *Create* component. This rule decides whether or not a location is new or it exists already. For this purpose, the potentially new location is checked against all already added locations. If there is at least one added location located within a `LOCATION_BUFFER` from the location under check, then the rule decides that the location already exists. Consequently, the *Create* component aborts the action of adding a new location and shows a warning to the user, as displayed in Figure 4.6. On the other hand, if the rule states that the location is indeed a new one, then it will be created. The `LOCATION_BUFFER` is a configurable parameter within

the *Create* component and was set to 10 meters for the purpose of this thesis.

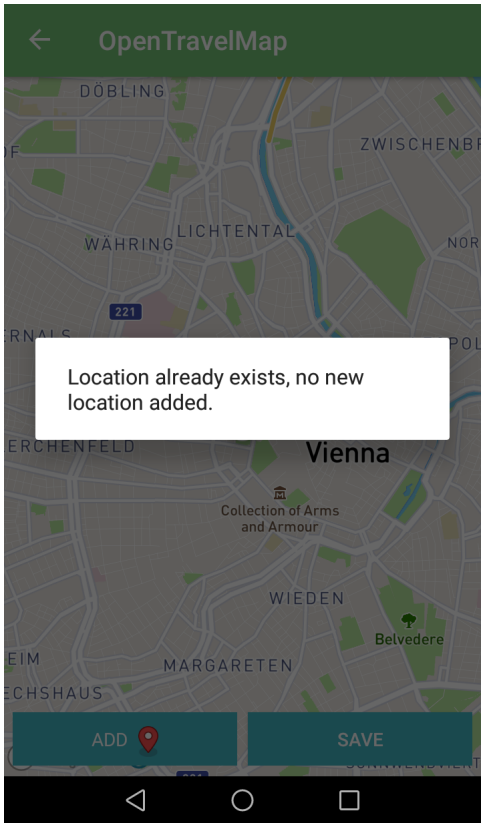

Figure 4.6: *Add existing location warning*

During the process of adding a location, the user is requested to provide a location name. In an attempt to assist the user in finding the most appropriate name, the *Create* component provides name suggestions as illustrated in Figure 5.3 left. This suggestions represent names of nearby locations which were already added, either by the current user during the creation of another route or by other users. The client's *Create* component asks the server for name suggestions by sending a request on the already mentioned "/getNearbyPoiNames" endpoint. The parameters of this request are the latitude and longitude coordinates of the location to add, together with a distance. The distance is used to perform a spatial query that will retrieve all names of locations located within this distance from the new location. Technically, the distance is defined as a configurable parameter within the *Create* component, the `DIST_TO_NEARBY_LOCATIONS` parameter with the value set to 250 meters.

Once the user added at least one location, the *Create* component automatically creates a route and enables the possibility to save the created elements. Within the saving process the route together with the created locations are sent to the server through a POST request on

the endpoint "/saveRoute". The client side of the platform waits for a response from the server and notifies the user of the success or failure of the save process, as shown in Figure 4.7. The save process might fail either due to problems occurring on the server side of the platform or because of an inexistent or interrupted client-server connection.

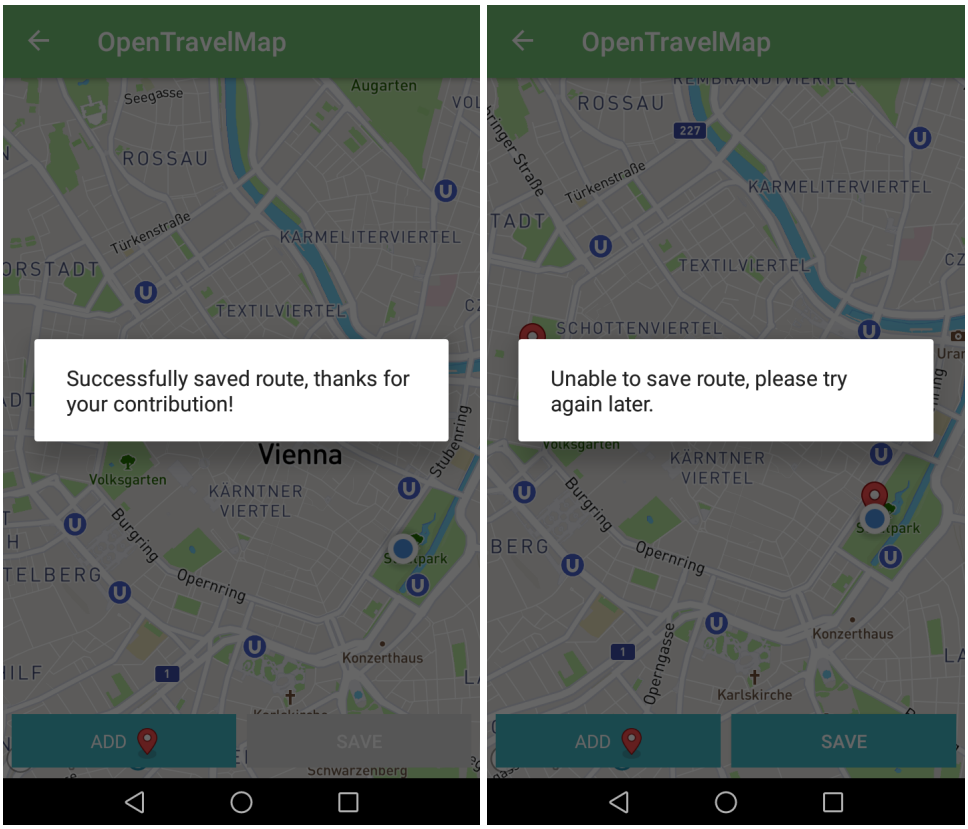

Figure 4.7: *Save route messages*

As already explained, the *Create* component gives the user the opportunity to create locations and once they were created, these locations can be edited and saved at any time in the future. After creating a location, the location will be automatically stored into the client's database and will remain there until the user decides to trigger the save process. On success of the save process the route and locations created within the *Create* component will be stored on the server's database and removed from the client's database. Each time the user accesses the *Create* component, the data from the client's database will be fetched. Until the save process is triggered and successfully finished, this data is not empty and the user will be notified with a proper message, as displayed in Figure 4.8. The user is informed that an unsaved route was found and is requested to decide whether or not the route will be kept or deleted. Keeping the route will provide further access to the created locations, enabling the edit and save functionalities. By contrast, if the user decides to delete the route then the

client's database will be cleared and the created data will be lost.

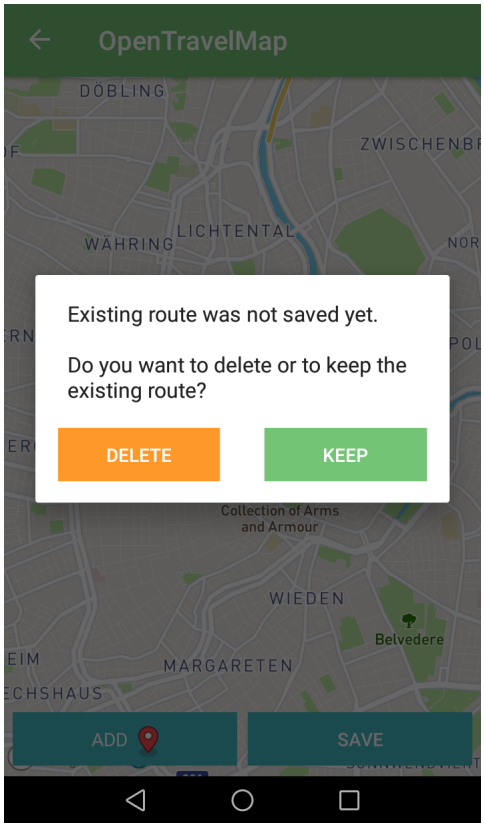

Figure 4.8: *Unsaved route message*

4.2.2 Search component

The setup of the *Search* component comprises mainly the behavior of the filter search functionality and the configuration of the recommender system. The basic functionality of the *Search* component assist the user in searching for travel routes, whereas the recommender system enhances the search result by considering the user's desires together with context related weather information.

During the search process, the *Search* component gains access to the routes created and saved within the *Create* component. All these routes, together with their corresponding locations, are stored on the server's database and will be sent to the client's *Search* component as response to the GET request sent on the endpoint "/getRoutes". As already explained, the server's database stores all routes created around the world from the beginning of the *OpenTravelMap* platform. At first, retrieving all routes from the database will not be a problem, but as the platform's popularity grows and the users start creating and saving more

and more routes, the size of the response to the "/getRoutes" request will rapidly increase. Moreover, it is clear from the first place that each user is interested in a particular location when the search process is performed. This location might be a country, region or city and the user will certainly focus on the search result located around this particular location. Consequently, a more efficient approach would be to filter the search result such that only routes located around the user's location of interest are retrieved. This approach is implemented within the *Search* component and illustrated in Figure 4.9.

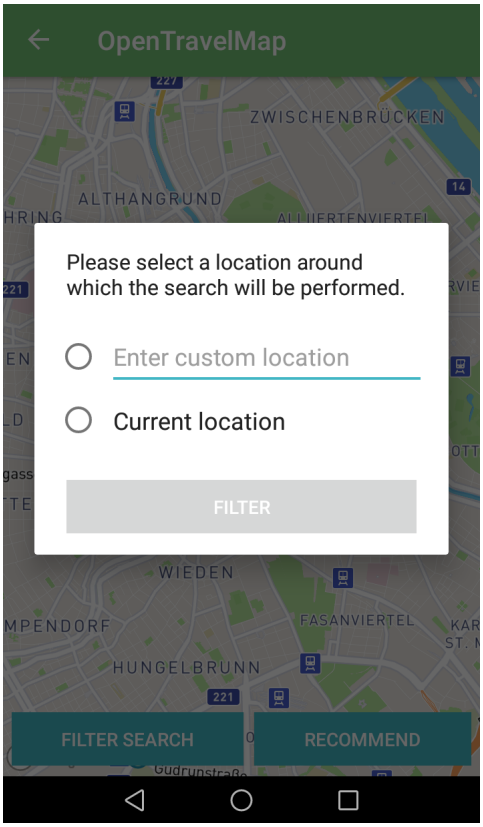

Figure 4.9: *Mandatory filter search*

As evident from the figure, filtering the search is a mandatory step, since the button with the "Filter" label is disabled as long as no user input is present. The user is required to enter a location around which the search process will be performed. This can be either the current location, which will be automatically requested from the device's location provider, or a custom location entered by the user. In both cases, the latitude and longitude coordinates of the entered location, together with a radius will be sent as parameters of the GET request on the "/getRoutes" endpoint. The server will perform a spatial query and retrieve only routes that contain locations located within the given radius from the entered location. The mentioned radius is stored on the client side of the *OpenTravelMap* platform, within the configurable parameter named `SEARCH_RADIUS`, whose value is set to 20 km for the purpose of this thesis.

Usually, after the filter is applied, the search process is performed and the search result is displayed to the user as shown in Figure 5.8. In order to avoid an unnecessary overload of the map, locations having the same name will be aggregated to their centroid and drawn only once on the map at the coordinates of this centroid. The coordinates of the centroid are calculated by averaging the latitudes and longitudes, respectively.

Nevertheless, it might happen that there is no search result to be shown. In such a case the user is notified accordingly, as indicated in Figure 4.10. The failure of the filter search process is illustrated on the left side of the figure. This can be caused by problems on the server side of the platform or an inexistent or interrupted server-client connection. On the other hand, in case of an empty search result, the user is warned with the message displayed on the right side of the figure. The cause of an empty search result is simply that there are no routes in the server's database located around the entered location.

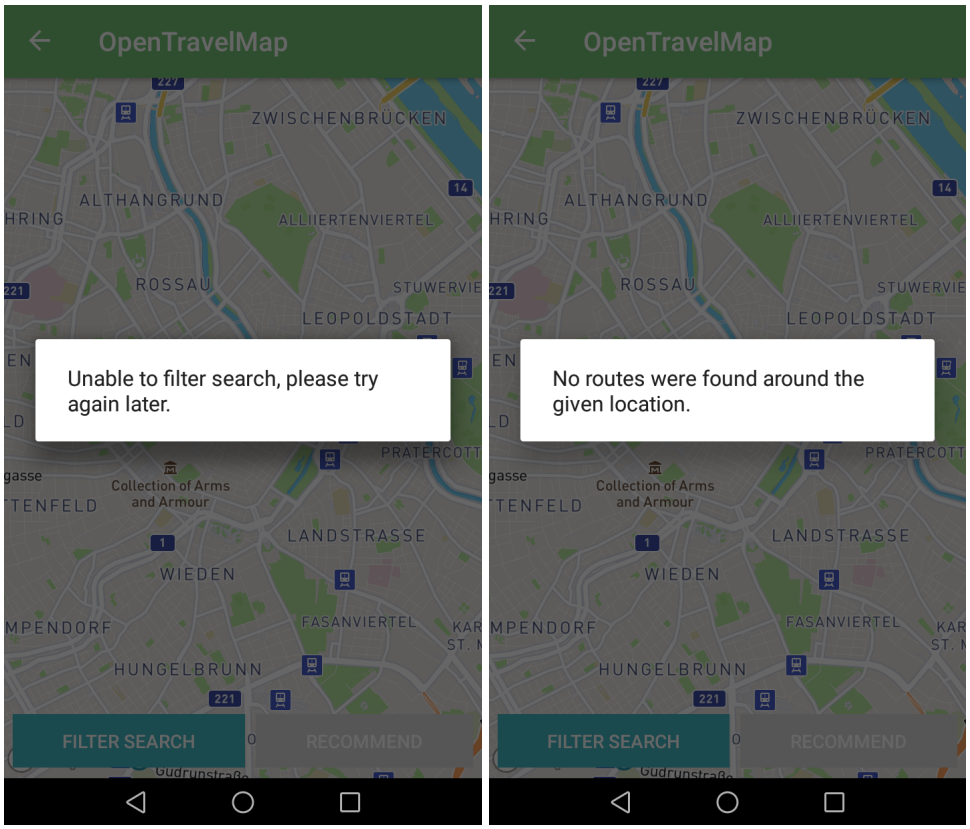

Figure 4.10: *Filter search warnings*

Beside the basic search functionality, the *Search* component also provides a feature that enhances the search result. This feature is a recommender system that was designed to take

the user's desires and the correspondent weather forecast into account, in order to retrieve recommended routes, built from locations that were part of the basic search result.

Illustrated in Figure 4.11 are the possible configurations of the recommender system. The user has the opportunity to set the type of the recommended routes by choosing from the options full-day or half-day, respectively. As expected, the chosen type will determine the total length of each recommended route, whereby the length represents the estimated time spent along the route. Moreover, the recommender system can also consider context related information if the user decides so. This information is represented by a weather forecast for the current day and location of interest. As indicated in Figure 4.11, the chosen parameters of each configuration are highlighted or written in blue. In addition, a proper weather icon is displayed when the weather option is enabled.

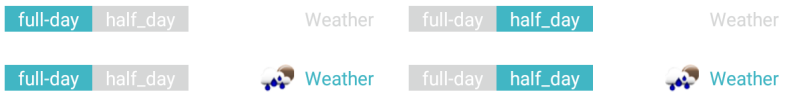

Figure 4.11: *Recommender configurations*

Depending on the chosen configuration, the recommender system creates new routes based on the locations retrieved within the basic search result. Each location is characterized, among others, by a location type which can be indoor, outdoor or both. In order to estimate the time spent along a route, the route was divided into its fundamental parts: locations and directions, whereby a direction is the distance between two locations. It has been assumed that per direction around 0.5 hours will be spent. Furthermore, for each location type the following time estimates were considered: indoor $\sim$ 2 hours, outdoor $\sim$ 0.5 hours, both $\sim$ 1.25 hours. It has also been planed that the user will take a break for about 1.5 hours to eat/drink. If the weather forecast has to be considered, then the recommender system receives the information whether or not rain is expected and filters the available locations such that in case of rain only locations of type indoor and both are used. For the purpose of varying the results, the recommender system first randomly shuffles the available locations and afterwards groups combinations of location types into new routes. A full-day route is designed as: *Location A + Location B + Break + Location C + Location D* and should not last longer than 8 hours, whereas a half-day route as: *Location A + Location B + Break* with a maximum of 5 hours time length. Depending on the chosen route type and the given weather information, the combinations of location types listed in Table 4.1 will be recommended. As explained earlier, the recommender system makes use of the searched locations and creates new routes. Normally, the user will be able to see the recommended routes in the view illustrated in Figure 5.11. However, if there are not enough locations such that at least one route can be created inside the recommender system, then the user will be notified immediately.

	no rain	rain
full-day	1. indoor + both + 2 x outdoor 2. indoor + 3 x outdoor 3. both + 3 x outdoor Fallback: 4 x outdoor	1. 2 x indoor 2. indoor + both 3. 3 x both
half-day	1. indoor + outdoor 2. both + outdoor 3. 2 x outdoor	1. indoor 2. 2 x both

Table 4.1: *Recommender system reasoning*

The corresponding warning message is displayed in Figure 4.12.

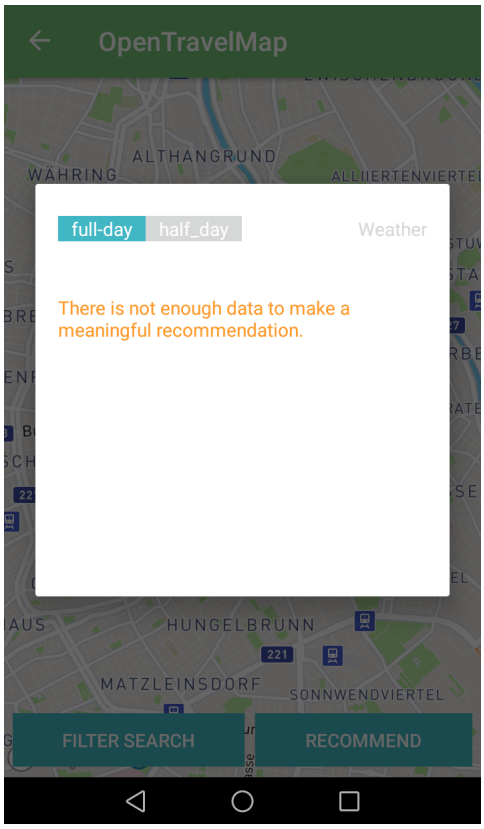

Figure 4.12: *Recommender warning*

Chapter 5

OpenTravelMap specification

The purpose of the *OpenTravelMap* platform is to provide user support in creating and searching for travel routes. In order to meet this purpose, the platform consists of two major components: the *Create* and the *Search* component, respectively. As the components names already suggest, the first component is responsible for creating travel routes, while the *Search* component covers the searching support within the platform. The separation of the *Create* and the *Search* component is done within the home screen of the platform as illustrated in Figure 5.1. The following sections provide a detailed description of both components.

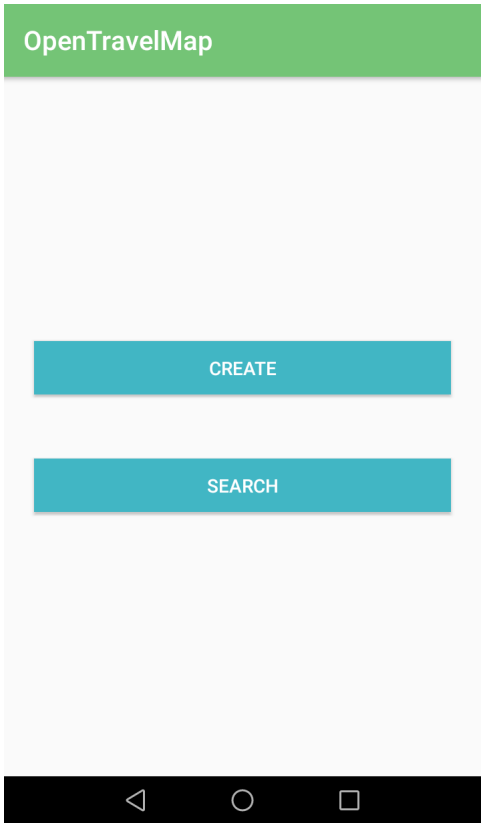

Figure 5.1: *Home screen*

5.1 Create component

The *Create* component represents the core functionality of the *OpenTravelMap* platform, since here is the place where the data is created. The aim of this component is to provide tools which assist users in the process of creating routes. A route consists of a sequence of locations and it is created when the first location is added.

The *Create* component allows the creation of one route at a time. Figure 5.2 shows the screen of the *Create* component with and without the GPS position of the current user, which is represented by a blue circle. As illustrated in the figure, there are only two actions that the user can perform: either to add a location (symbolized by a red marker) or to save the route. At this point no location was added, consequently there is no route, which is why the save button is disabled.

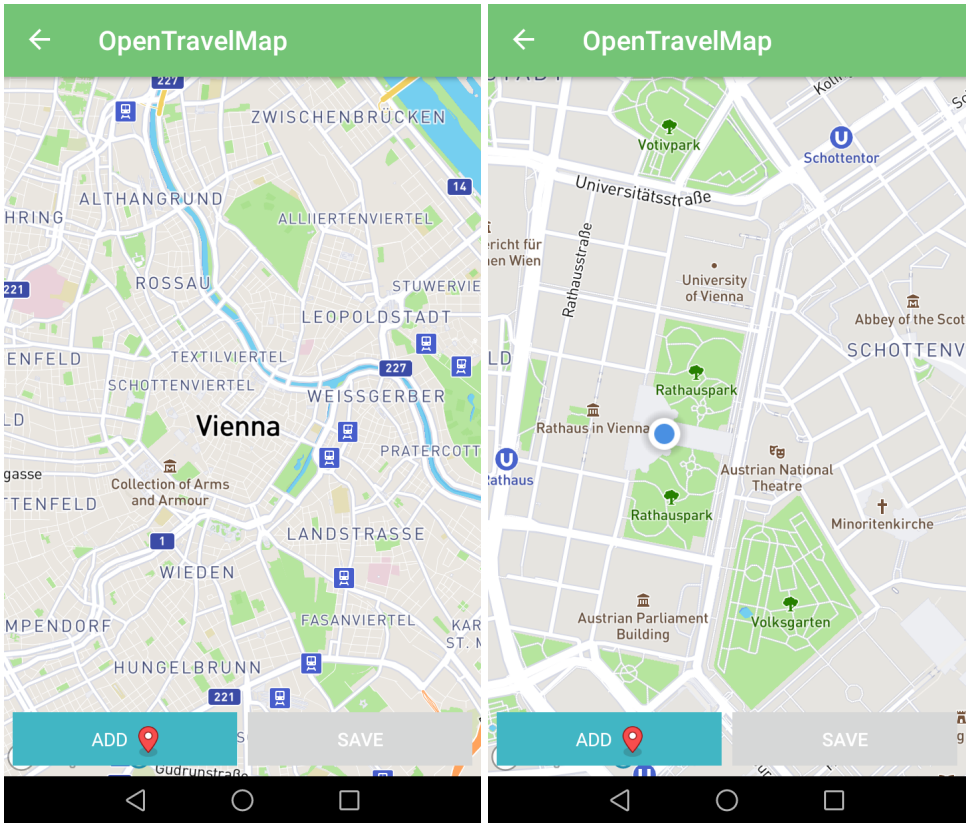

Figure 5.2: *Create screen*

Within the *Create* component there is a significant difference between the following two actions which can be applied to a route: create and save. The creation of a route takes place on the client side of the platform and it is automatically triggered, as mentioned before,

when the first location is added. By contrast, the save route action is triggered by pressing the save button which will request the server side of the platform to archive the route and its corresponding locations.

In consequence, the first step that can be performed in the *Create* component is the creation of at least one location. A location can be added/created by pressing the button located on the left side of the screen illustrated in Figure 5.2. Each location will have coordinates assigned to it, which are the exact latitude and longitude GPS coordinates of the user's device. Therefore, a crucial requirement for adding a location is enabling the GPS of the user's device. Once the current position of the user's device is provided then a location can be added.

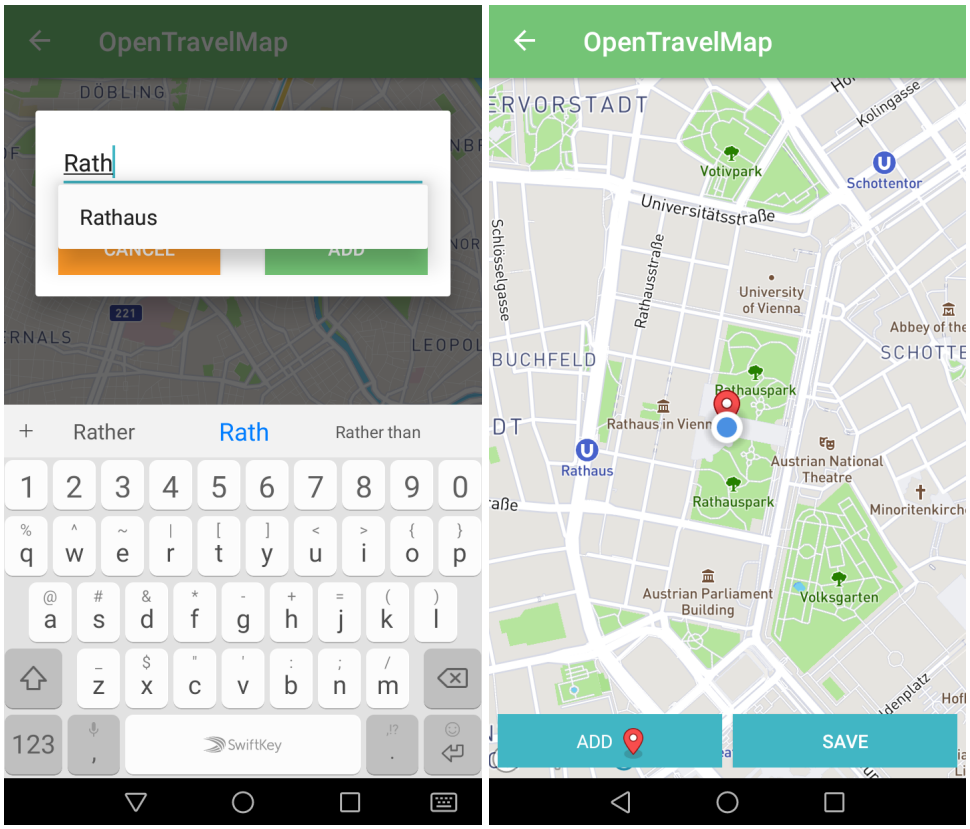

Figure 5.3: *Add location view (left), Location added (right)*

The process of adding a location is depicted in Figure 5.3. As shown on the left side of the figure, the user is requested to provide a name for the future location. In order to assist the user in choosing a suitable name several suggestions pop up. This suggestions represent names of nearby locations which were already added, either by the current user during the creation of another route or by other users. The location name is mandatory, therefore the user has to choose one of the suggested names or provide a new one.

The location name is an essential element that is used not only across the *OpenTravelMap* platform but also send as a request parameter to external apis, in order to obtain further information which is relevant for the added location. Therefore, users are encouraged to provide valid and meaningful location names in an attempt to take full advantage of the functionalities covered by the platform.

Comparing the right sides of Figure 5.2 and Figure 5.3 shows that a location was added. The added location is displayed on the map as a red marker pinned at the current position of the user. Furthermore, the save button is now enabled, which confirms the fact that a route was also created and from now on can be saved. Even though the save functionality is already available, this does not mean that the user should use it and therefore end the process of creation immediately.

Figure 5.4 illustrates another feature of the *Create* component, namely the user's opportunity to edit an added location. The edit tool consists of a view which appears once the

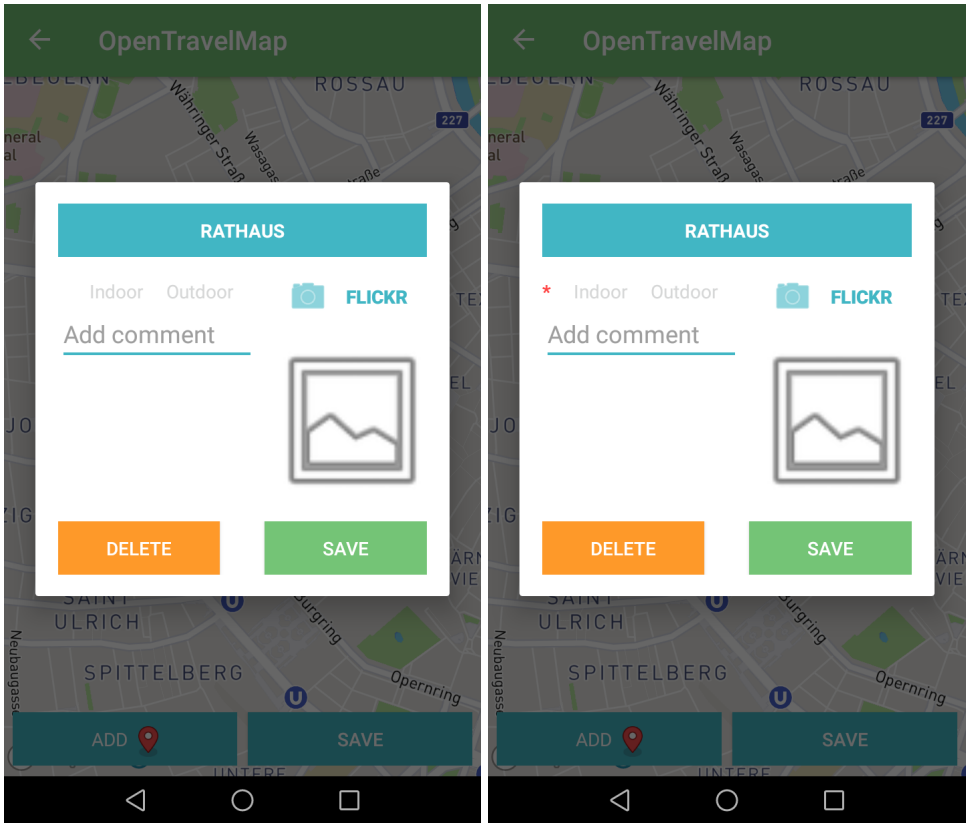

Figure 5.4: *Edit location view (left), Edit location constraints (right)*

user pressed the marker of the added location. As displayed in the figure, the location name

is shown in the header of the edit tool. The body of this view comprises several attributes that can or need to be assigned to the location. At the bottom of the view there are two buttons which give the user the possibility to either cancel or save the changes. In contrast to the location name view from Figure 5.3 left, the edit location view is cancelable, meaning that the user is allowed to press anywhere outside the view and the view will disappear, causing the loss of any changes made inside it. Hence, the only way to guarantee that the changes will be preserved is to press the save button on completion. This will update the saved instance of the location on the client side of the platform making it accessible also later on, until the entire route is saved.

As already mentioned, beside the name, latitude and longitude, a location can also have several attributes which are shown in Figure 5.4. The left side of the figure displays the initial edit view, whereas the right side shows the edit view after the save button was pressed. Comparing the images shows that the location type (indoor and/or outdoor) is the only mandatory attribute. Evidence for this is the red asterisk which appears on the left side of the edit view, right beside the location type choices. Even though the location type is a mandatory attribute, there are still cases in which a location will not have this attribute set. This can happen when a route is saved an one or more of its locations were either not edited at all or their corresponding edit view was canceled. The location type attribute plays a major role within the *Search* component, hence, for all locations that do not have the location type set, the outdoor location type will be further used.

Apart from that, the user can add a comment and/or a picture to each location. The comment can be introduced in the input text field that holds the hint "Add comment", while the picture can either be taken with the device's camera or requested from *Flickr*. The provided picture will be displayed above the image placeholder. The user is encouraged to contribute as much as possible since the output of the *Create* component represents the input of the *Search* component. Therefore, the more and detailed the created data is, the better and useful the search tools will be.

As specified, the *Create* component offers two possibilities to add a picture to a location within the edit view. The first option is to take a picture with the device's camera. Pressing the camera icon displayed on the right side of the edit view, above the image placeholder will open the image capture view. As soon as the user accepts the captured picture, this will be assigned to the location and replace the empty image placeholder. Another way to add a picture to a location is to choose one from a provider. For the aim of this thesis a connection with the *Flickr* platform was established. Pressing the "Flickr" button, located on the right side of the camera icon will send a request to the *Flickr* platform. The platform is requested

to send back public images taken around the location and also related to the location, since the name of the location is also send as a request parameter. In case of success, *Flickr* responds with a series of pictures.

The collection of Flickr images is further reduced to a limited number of choices which are shown in a scroll view as illustrated in Figure 5.5 left. Now the user has the opportunity to choose one of the images by simply selecting it. Essentially the same as in the case of taking a picture with the device's camera, the selected image will be attached to the location and shown in the edit view. In an ideal case, the user has provided not only the mandatory

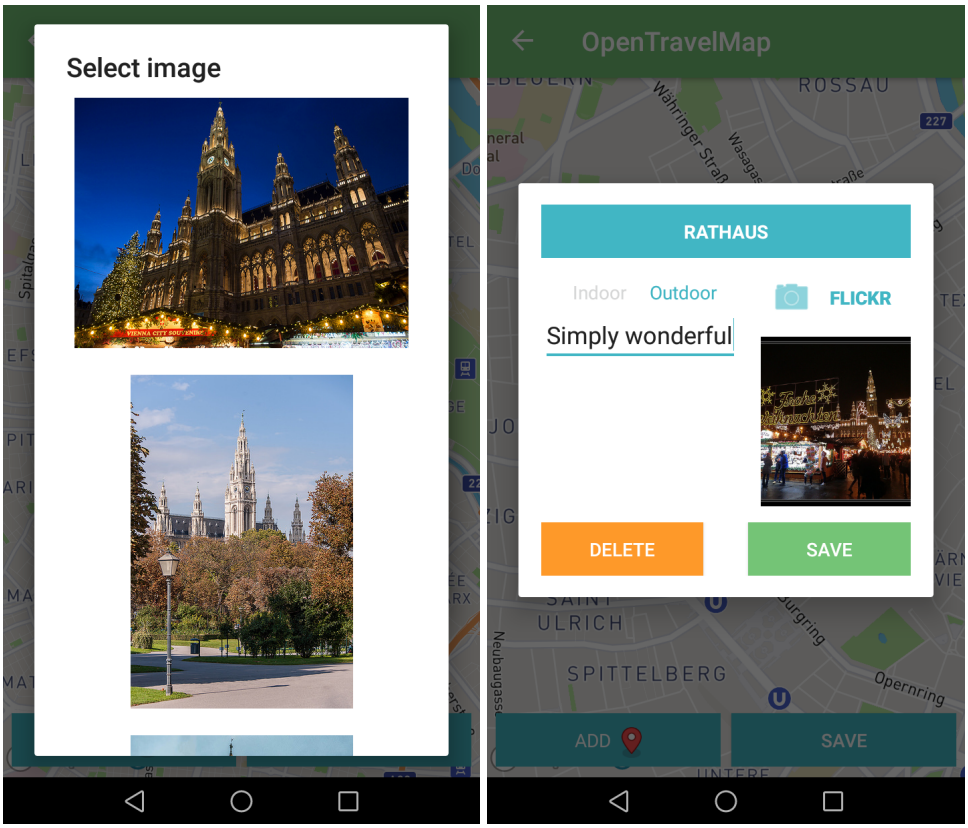

Figure 5.5: *Flickr images view (left), Edit location filled (right)*

location type but also all the other optional attributes. Thus a detailed description of the location was created. Displayed on the right side of Figure 5.5 is such an ideal case. In addition to the outdoor type which is assigned to the added location, the location also includes a comment and a picture. The final step that is required in order to preserve the created content is to save the changes. This will save the data on the client side of the *OpenTravelMap* platform making it available for further use.

As explained before, within the *Create* component the user can perform two main actions:

add a location and save the created route. A route consists of at least one location. By taking this definition into account, it is obvious that as soon as one location was created the route can be saved. However, this does not characterize a real scenario, since normally a route would comprise more than one location. Such a real case scenario presenting a route with three locations is illustrated on the left side of Figure 5.6. The added locations are shown on the map as the three red markers, together with the position of the user which is displayed as a blue circle. By pressing each of the red markers the edit view of the corresponding location will appear giving the user the chance to add or change the location attributes described above.

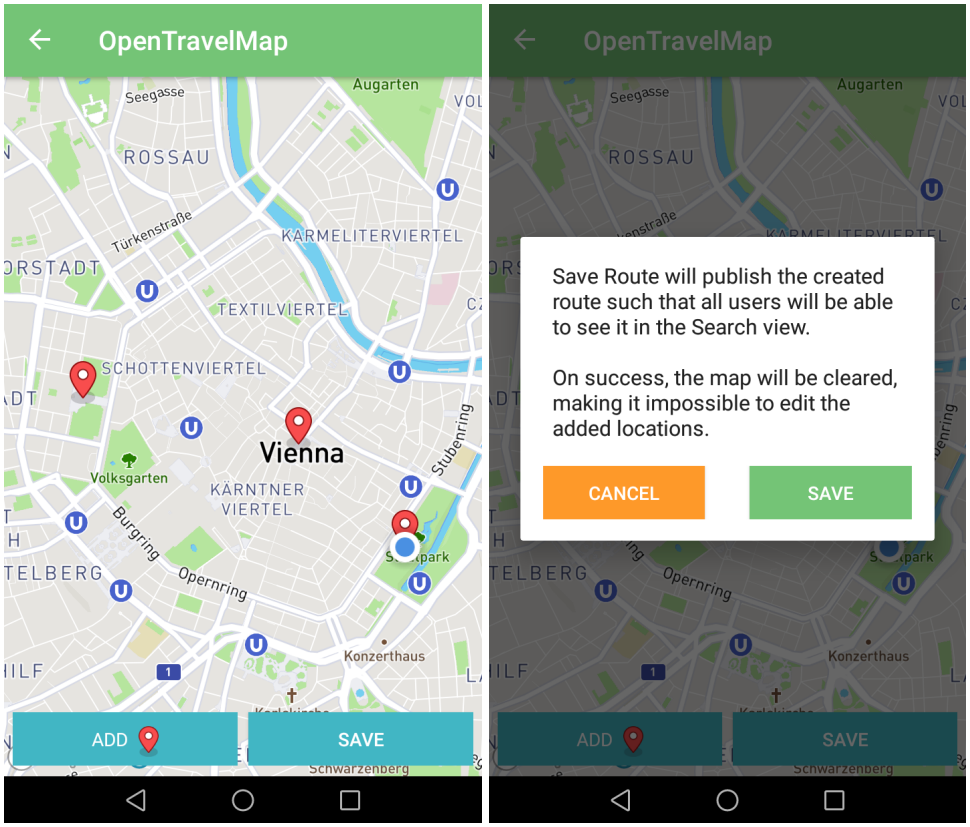

Figure 5.6: *Save route*

Due to the fact that the process of saving a route has irreversible effects on the current state of the created content, it has been designed to be performed in two steps. First the save button from the *Create* component view (see Figure 5.6 left) has to be pressed. After pressing this save button a new view appears, see Figure 5.6 right. This new view represents the second and last step of the save route process. The view shows an explanation of the save route process and requests the user's decision to either continue or cancel the save process. Within the save process the created route and its corresponding locations are

sent to the server side of the *OpenTravelMap* platform where they will be stored. On success, the map will be cleared, causing all the red markers to be deleted. Furthermore, the route and locations stored on the client side of the platform will also be deleted. Consequently, the user will not be able to change the added locations anymore. By choosing to continue the save process and if it will successfully complete, then the actions described above will take place. However, if the user chooses to cancel the save process or if the process does not complete successfully, then the map and the client side storage will preserve their current states. The added locations will still be stored on the client side of the platform, displayed as red markers on the map and open to changes. The user will then have the opportunity to save the route each time in the future.

As described in the previous chapter, the user will be notified both on success and on failure of the save route process. The corresponding messages are shown in Figure 4.7. Beside the mentioned actions that will take place in case the save process is performed, there is one more effect that has to be clarified to the user. Once the route and its corresponding locations are stored on the server side of the platform the user's contribution is made public through the *Search* component, which now has direct access to them. In other words, the output of the *Create* component is used as input in the *Search* component.

5.2 Search component

The goals of the *Search* component are on the one hand to enable content sharing between users and on the other hand to enhance the route search, if requested, by offering recommendations that adjust the retrieved content through context awareness. The *Search* component was designed to make use of the data gathered with the support of the *Create* component. Therefore, it serves as a tool to publish content by making it available to any interested user. The second feature of the component is its ability to provide relevant route recommendations to its users.

The basic functionality of the *Search* component is to provide support in searching for travel routes. In order to achieve this, the component requests routes from the server side of the platform. These routes were previously created and saved within the *Create* component. Even though this basic functionality seems straight forward, there is an essential issue that has to be addressed. By default, the request send to the server side of the platform will cause a response containing all the stored routes. In a real case scenario the user is not interested in all available routes but just in routes surrounding a certain location, for example a city. One way to restrict the retrieved content is to apply a filter during the request. The *Search* component uses this method which is illustrated in Figure 5.7.

As soon as the user pressed the search button from the home screen of the *OpenTravelMap* (see Figure 5.1), the filter search dialog shown in Figure 5.7 appears. Within this step the user is required to choose the way in which the search will be filtered. As indicated in the figure, there are only two possibilities to choose from.

The first option is that the user enters a custom location. As evident from Figure 5.7 left, the *Search* component assists the user by offering suggestions that match the entered value. It is of great importance that the user provides a valid custom location, such as the ones being suggested. The reason for that is that the entered location must be geocoded in order to perform the search. During the geocoding process, the value entered is transformed into latitude and longitude coordinates, describing the location provided. Therefore, wrong or misspelled custom locations will cause a failure of the geocoding process. Consequently, the search process will also fail.

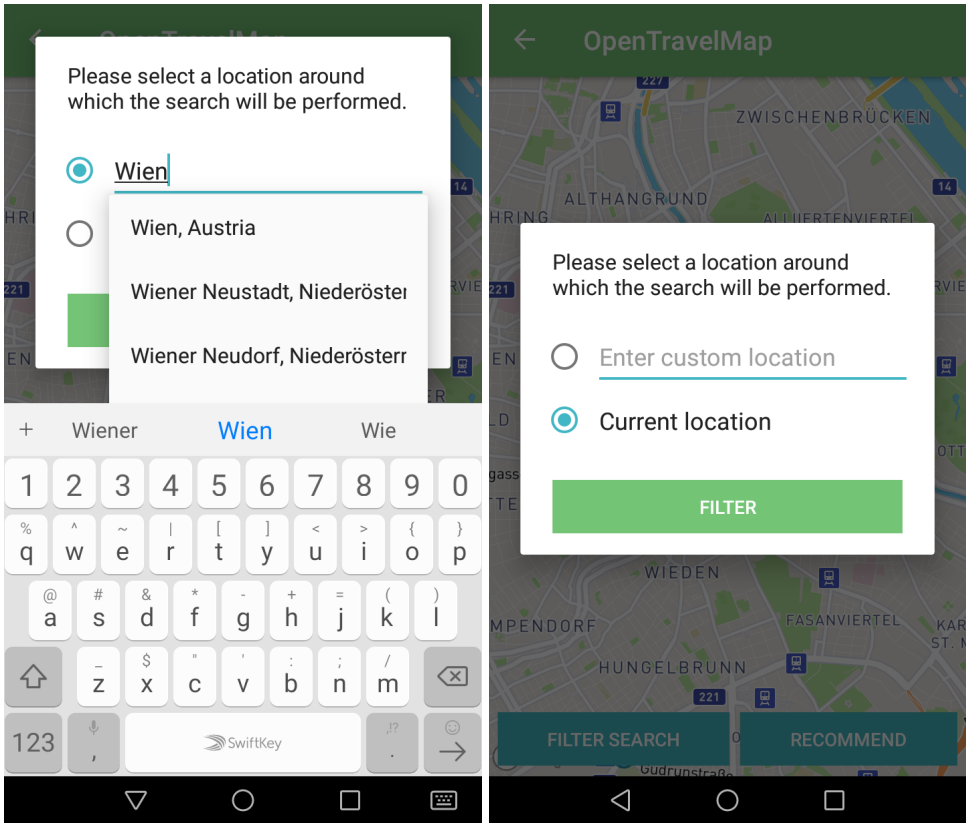

Figure 5.7: *Filter search*

The second possibility is to use the current location of the user as shown in Figure 5.7 right. This assumes that the device's location provider is enabled and tries to access its current

position. It is recommended that the GPS is enabled during the usage of the *Search* component, but it is not mandatory like for the *Create* component. In case the GPS is disabled or when there was no valid position provided, then the search will fail. Another reason that could cause the failure of the search process is the missing or interrupted connection with the server side of the platform, since all the data is stored and queried there. If due to some of the mentioned causes the search process fails, then the user will be informed as illustrated in Figure 4.10 left. On the other hand, if the search completes successfully but there were no routes found around the provided location, the user will get the message displayed on the right side of Figure 4.10.

The result of a successful search is given in Figure 5.8. It consists of a collection of routes that contain locations which are placed around the location entered in the filter search step. In order to prevent an overload of the view, the directions between locations are not shown on the map. The only elements that are shown are the locations, which are displayed exactly the same as in the *Create* component, as red markers. As already mentioned, the blue circle represents the current location of the user.

Figure 5.8: *Result of filter search*

Apart from the search result, two buttons can be observed at the bottom of Figure 5.8. The button placed on the left side of the screen opens the known filter search view, which is displayed in Figure 5.7. A feature of the *Search* component hides behind the recommend button, which is located on the bottom right side of the screen. The capabilities of this feature will be discussed later on this chapter.

This filter search button offers the user the possibility to further filter the search even after accessing the *Search* component. The only difference between the filter search view that appears when trying to access the *Search* component and the one that is displayed after pressing the button inside the component is the fact that the second view is cancelable. In other words, entering a location around which the search will be performed is mandatory when accessing the *Search* component and optional afterwards.

Once the search was performed and the results were shown, the user has the chance to explore the map, view detailed information about the displayed locations, the directions connecting them and the routes they belong to. For the purpose of accessing more information about a certain location, the user has to press the red marker symbolizing the location in the map view. By pressing the location's marker, the view illustrated in Figure 5.9 left will pop up. This view looks similar to the edit location view displayed in Figure 5.5 right, but none of the information displayed here is editable. As can be seen in the figure, the header of the view contains the name of the location. Below the header, there are two scrollable elements, the first one showing the location's pictures and the second one listing the comments added to the present location. All this information originates from the *Create* component where it was created and saved such that now it is publicly available within the *Search* component. The two buttons at the bottom of the view give the user the possibility to perform further actions such as to search for places to eat near the viewed location or to display all the routes that contain the present location. For the moment only the second action will be discussed.

As already mentioned, within the initial view of the search result shown in Figure 5.8 the directions connecting the locations and therefore the routes to which they belong to are not displayed in order to avoid an overload of the map view. The only way to show this information on the map is to access it through a location. This is done by pressing the button located on the bottom right side of the view displayed in Figure 5.9 left. After pressing this button, the view containing detailed information about the selected location will disappear. Furthermore, all routes containing the previous selected location will be shown on the map as indicated in Figure 5.9 right.

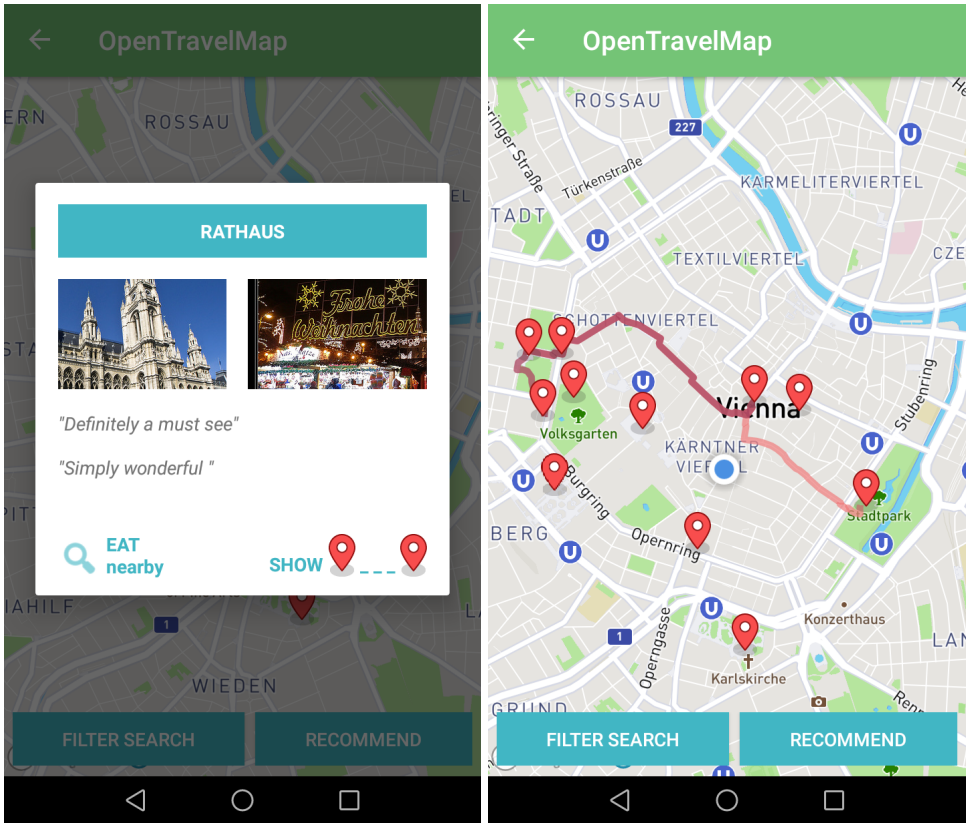

Figure 5.9: *View location (left), View routes passing by location (right)*

The right side of Figure 5.9 illustrates the map view containing all the locations of the search result and two routes. The routes are displayed on the map in different colors, making them easily distinguishable. Each route consists of locations and directions which connect those locations. Each direction has exactly two locations, a start and an end, and it is drawn on the map in the same color as its corresponding route. In case two or more routes overlap in certain regions, then these regions will be displayed in a mixed color formed from the colors of the overlapping routes.

Similar to pressing one of the red markers on the map in order to obtain detailed information about one of the displayed locations, also pressing one of the colored lines on the map offers more information to the user about the selected direction. The view that appears after the user selected a direction from the map is shown in Figure 5.10. The left side of the figure shows a view containing information about a direction that belongs to a single route, whereas the right side of the figure details a directions which is part of two different routes.

The header of the view shown in Figure 5.10 contains the names of the two locations corresponding to the selected direction. Below the header, there is a field that indicates the

distance in kilometers of the chosen direction. Under the "Overview" label there is a scrollable element, inside which all the routes containing the present direction are listed. Each route is listed as a sequence of locations having the exact same color as the one used to draw the route on the map. Apart from that, also the total distance of the route is revealed.

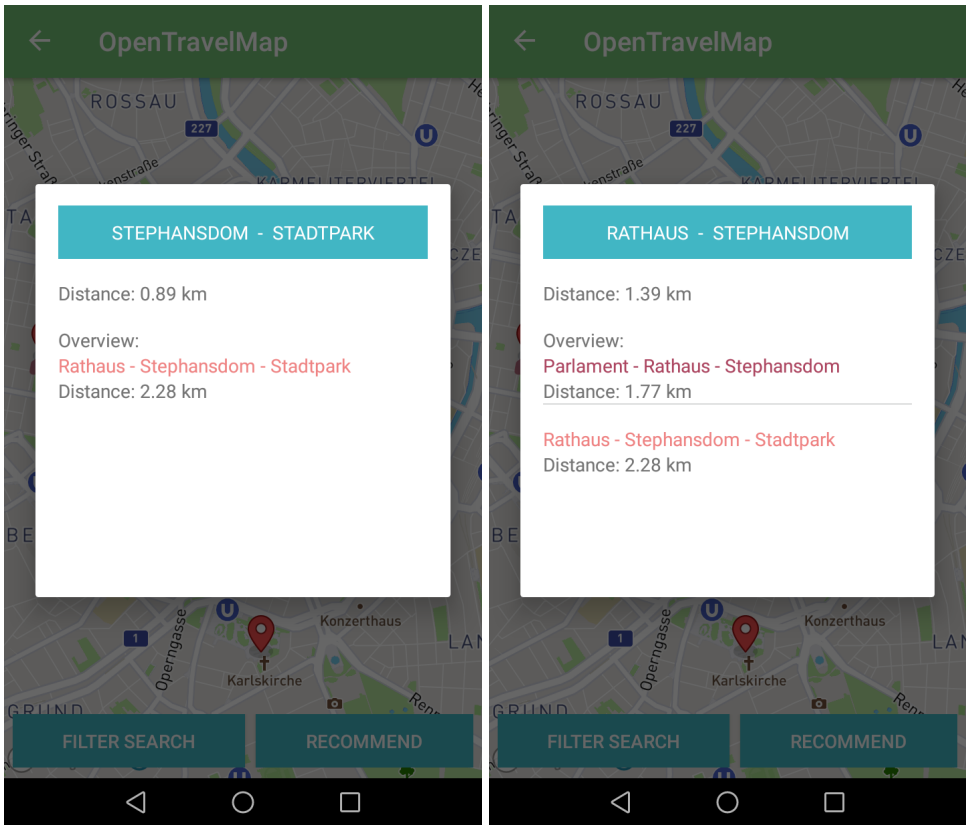

Figure 5.10: *View directions*

Until now, only the basic functionality of the *Search* component was covered, but the component also possesses a feature that enhances the basic search results. This feature is a recommender system that also feeds with routes originating from the *Create* component, just like the basic functionality. However, the recommender system adapts the search results by applying different filters and combining the retrieved locations such that also context information is taken into account. In order to access the recommender system, the user has to press the button labeled with "Recommend", which is located on the right side of the screen illustrated in Figure 5.8.

Once the user accessed the recommender system, a view similar to the one indicated in Figure 5.11 appears. This view contains both the setup as well as the results of the recommender. The setup is located inside an editable panel on the top of the view. As already

explained, there are four different setups that can be used to create recommendations. All these setups are displayed in Figure 4.11.

The user can edit two fundamental parameters inside the recommender system: the type of the resulted routes (categorized by length into: full-day or half-day) and the consideration of the current weather conditions. Figure 5.11 shows two examples of recommendations obtained under different setups. On the left side of the figure the recommender is set to full-day routes and no weather check, whereas on the right side to half-day routes with weather conditions taken into account.

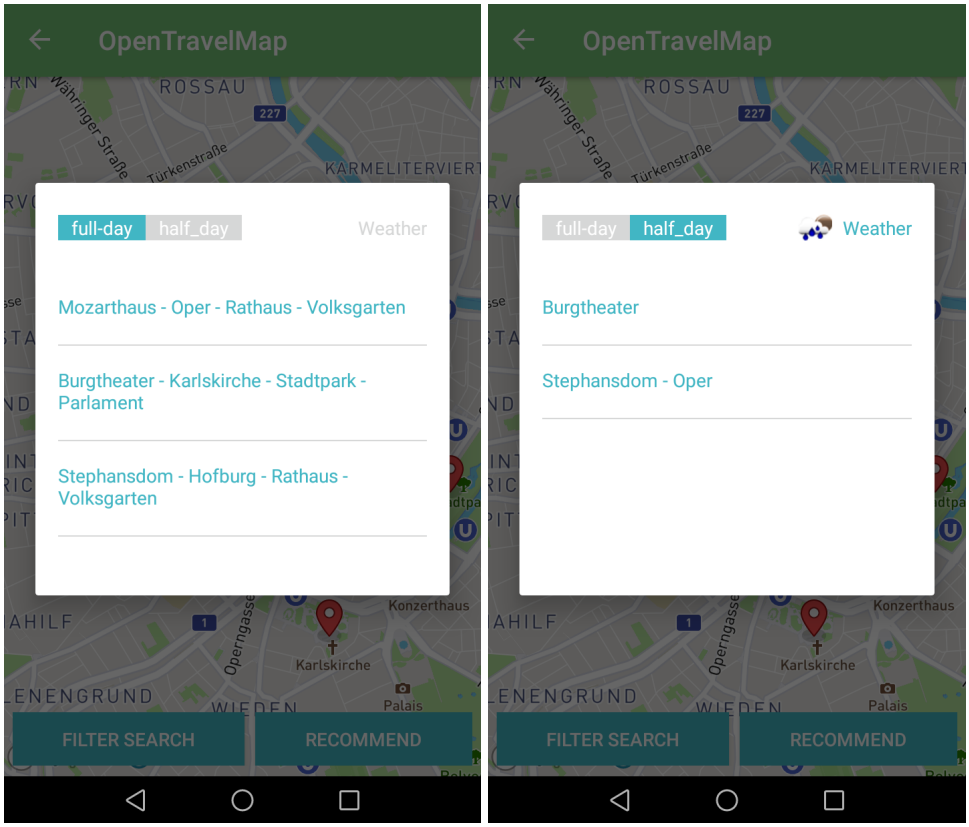

Figure 5.11: *Recommendations for: full-day with no weather (left), half-day with weather (right)*

The recommender system retrieves routes containing locations which were created and saved inside the *Create* component. The difference between the basic functionality of the *Search* component and the recommender is that the latter one does not preserve the routes originating from the *Create* component but forms new routes by combining the existing locations according to the setup provided by the user. These new formed routes represent the outcome of the recommender system, the recommendations. As indicated in Figure 5.11, the recommendations are listed below the setup as sequences of locations. As soon as the user

changes the setup, the recommendations view will be immediately updated.

Another significant difference between the recommender system and the basic functionality of the *Search* component is the way in which routes are listed. As shown in Figure 5.10, the basic functionality displays the routes in a variety of colors, each listed sequence of locations having the color in which its corresponding route is drawn on the map. By contrast, inside the recommender system all routes are listed in the same color, see Figure 5.11. The reason for that is on the one hand that the routes will not be drawn on the map, on the other hand because each sequence of locations is actually a button. Once the user pressed on one of the recommended routes, the view illustrated in Figure 5.12 will be shown.

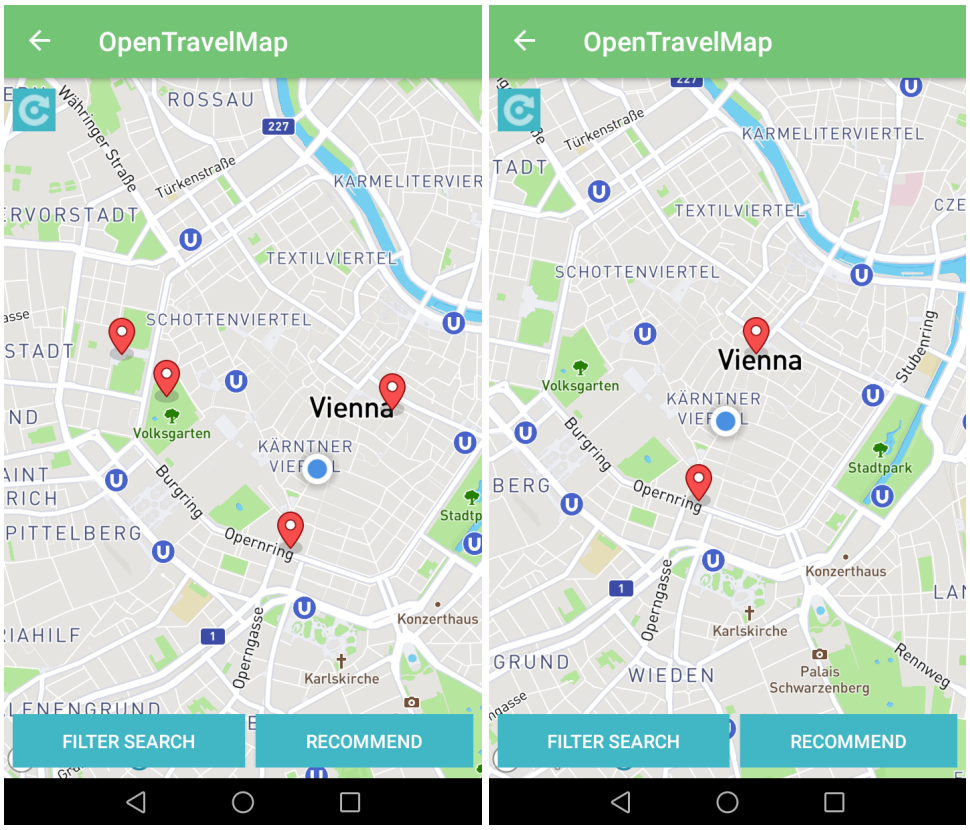

Figure 5.12: *Chosen recommendations for: full-day with no weather (left), half-day with weather (right)*

Even though Figure 5.12 looks similar to Figure 5.8, there is a remarkable difference between them. By comparing the mentioned figures one instantly notices that in Figure 5.12 there are fewer locations displayed on the map. That is because this figure shows only one recommended route, whereas in Figure 5.8 all routes are illustrated. The shown recommended route is exactly the one that was previously selected by the user inside the recommender

view. In addition, in Figure 5.12 there is also a refresh button displayed. This button is located on the top left side of the screen and provides the user the possibility to refresh the map view such that the view from Figure 5.8 is restored.

As already mentioned, a recommended route will not be fully displayed on the map. While the locations corresponding to the route will appear as red markers on the map, the directions connecting them will not be drawn. Thus, the user has the choice to decide in which order the locations will be visited and also which directions will be followed. In order to assist the user during travel, the *Search* component also displays the current position of the user as a blue circle on the map. However, this only happens when the device's location provider is enabled.

Now the user's mission started and its objective is to visit all locations of the recommended route. Further information about the recommended locations can be accessed by pressing the red markers on the map. The view that appears after pressing one of the red markers is displayed on the left side of Figure 5.13. As expected, the view is almost exactly the same

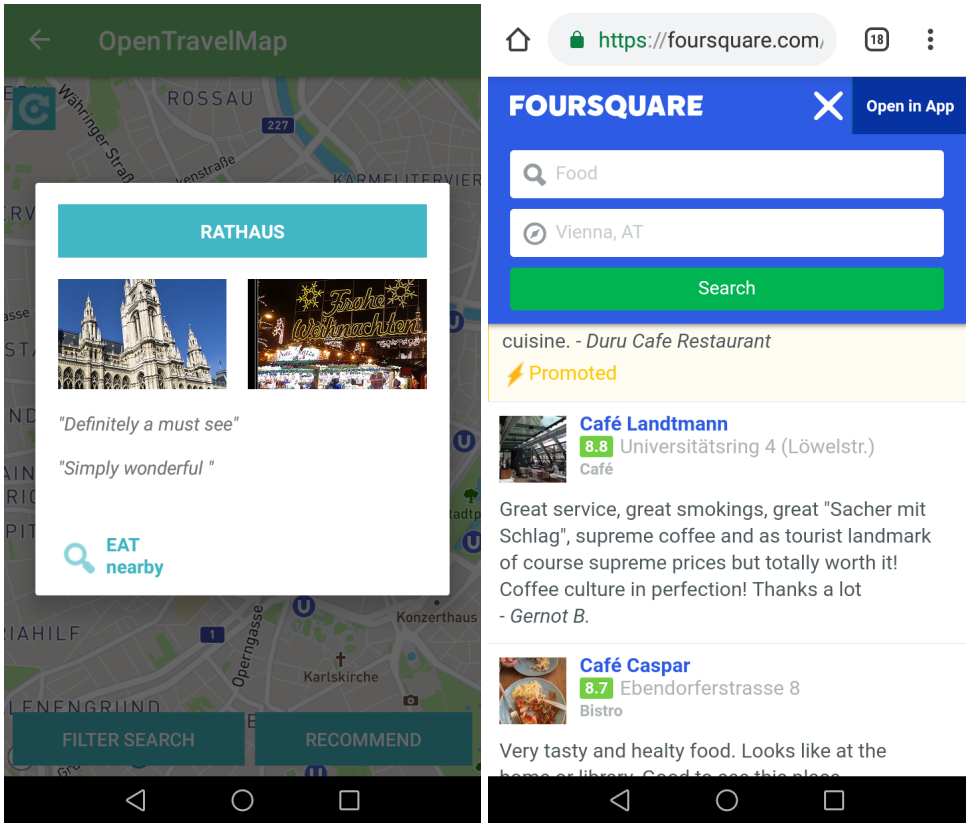

Figure 5.13: *View recommended location (left), Eat nearby view (right)*

as the one illustrated in Figure 5.9 left. The remarkable difference is the absence of the show

directions button on the bottom right side of the view. This button is not shown, since there is no support for this feature in the context of a recommended route. However, of great importance in this context is the button located on the bottom left side of the view. The button labeled with "Eat nearby" hosts a link to the *Foursquare* platform. After pressing this button, the user lands on the page displayed on the right side of Figure 5.13. This page contains a collection of places to eat/drink, all located near the location viewed by the user. The user can now browse through the collection of places and explore the features provided by the *Foursquare* platform. By pressing the back button, the user returns to the *OpenTravelMap* platform, more specifically to the view illustrated on the left side of Figure 5.13.

Chapter 6

Results and interpretation

The purpose of this thesis was to develop the *OpenTravelMap* platform, a platform that enables and promotes the involvement in volunteered geographic information. The *OpenTravelMap* platform offers assistance in the process of creating and searching for travel routes, while providing the following features: components for route creation and search in one platform, context aware recommendations and integration with other platforms in order to access existing resources.

In contrast to other existing platforms, the *OpenTravelMap* not only supports both the possibility to create and search for routes, but also aims to enhance both functionalities. The *Create* component provides a way to create routes and to describe the added content. Furthermore, the *Search* component of the *OpenTravelMap* platform allows sharing information between users and also adapts the retrieved content through context awareness.

The outcome of this thesis is a mobile location based service delivered through the *OpenTravelMap* platform. The platform was developed as a client-server application running on mobile Android devices. The main objective within the development of the platform was to integrate tools to improve the basic functionality while keeping the design of the user interface as simple as possible. The basic functionality of the *OpenTravelMap* platform offers support for creating routes as collections of locations and enables the possibility to search for travel routes. On top of this, there are features that enhance both the *Create* component and the *Search* component, respectively. The enhanced *Create* component supports editable locations and direct access to external resources, whereas the improved *Search* component offers context aware recommendations that are based on the basic search results but adapted to meet the user's preferences and to consider relevant weather information.

The main target audience of the *OpenTravelMap* platform consists of tourists. Therefore, an essential requirement within the development of the platform was the ability to run on mobile devices. This way, the target users that are usually under continuous movement gain immediate access to the features of the *OpenTravelMap* platform. A major constraint

that spreads along the *OpenTravelMap* platform is the dependency on an established internet connection. Even though having internet access on mobile devices should not be a problem nowadays, its understandable that this could be a reason why some users will avoid using the features of the *OpenTravelMap* platform.

A great improvement of the basic search functionality of the *OpenTravelMap* platform is represented by the integration of the context aware recommender system within the *Search* component. Due to the fact that the current capabilities of the recommender system are slightly limited, the enhancement of the recommender system is one of the topics suggested to be explored within the scope of future work.

All in all, the *OpenTravelMap* platform offers support both for route creation as well as for route search, thus overcoming the gaps existing in similar platforms. Moreover, the platform also incorporates a context aware recommender system which helps users in the process of decision making.

Chapter 7

Improvements and future work

The current state of the *OpenTravelMap* platform reveals a functional mobile location based service that assists users in creating and searching for travel routes. The main components of the platform comprise both tools that support the primitive create and search functionalities as well as advanced features that enhance the basic capabilities of the service. This chapter aims to provide suggestions for potential improvements that may extend the present *Create* and *Search* components of the *OpenTravelMap* platform.

In an attempt to enhance the *Create* component, future work could focus on handling the directions between locations. For the moment, the locations play the major role within the *Create* component, as the user is allowed to create and edit only those elements. Nevertheless, a route is composed of locations and directions that connect these locations. Since the path that a user follows from one location to another contains relevant information that might be worth sharing, future enhancements could consider introducing directions as additional elements within the *Create* component.

Another essential improvement relates to the process of adding a new location. Currently, the entire *OpenTravelMap* platform relies on the fact that the user provides meaningful names for the added locations. Even so, if the same location is added multiple times with the same name translated in different languages, the platform will not be able to match the names and unify the locations within the search process. Therefore, further work should investigate proper methods that could be applied to solve this issue.

Concerning the *Search* component, candidates for improvement are the visualization of the search results and the context aware recommender system. For the beginning the current approach, which shows only the locations of the search result and on demand also the routes passing by a certain location is enough to avoid the overload of the map. However, with growing popularity, the number of retrieved locations will increase rapidly and the visualization of all locations of the search result will deteriorate the usability of the *Search* component. The same issue will occur also when displaying the details of a certain location,

due to the high number of pictures and comments assigned to that location. Consequently, future work should consider applying techniques that will either limit the number of locations, pictures and comments or find a way to adjust the visualization of these elements such that the usability is not distorted. One approach to overcome this issues would be to connect the *OpenTravelMap* platform with social media platforms. Within this approach the search result would be filtered such that the retrieved locations would be the ones either created or visited by the user's friends from the social media platforms.

Further developments of the *Search* component could focus on the enhancement of the incorporated recommender system. Potential improvements may concern both the recommendation process itself as well as the visualization of the recommended routes. Related to the recommendation process, future work may implement methods that learn the user's preferences over time and adjust the recommendations accordingly. In order to achieve that, the platform should integrate user profiles. Ideally these profiles would be generated automatically by monitoring the user's activity within the major components of the platform. Created routes comprised of liked and disliked locations as well as accepted and rejected recommendations contain valuable information that should be stored within the user's profile. Concerning the visualization of the recommended routes, optimization techniques may be used to connect the route's locations, while considering further relevant aspects such as the opening hours of points of interest.

In future, improvements related to the entire *OpenTravelMap* platform should consider deploying the location based service on various mobile operating systems. Currently, the platform runs only on Android devices, but within the scope of future work it could also be developed for iPhone and Windows Mobile devices.

Chapter 8

Summary and conclusion

The structure of the present thesis was conceived to first perform a detailed analysis of related work released on the market. Afterwards, theoretical principles that influenced the development of the *OpenTravelMap* platform were discussed. Comprised within the current thesis were also technical insights related to the architecture and development methods of the mentioned platform. Furthermore, a comprehensive specification of the features provided by the *OpenTravelMap* platform was outlined. The results and achievements of this thesis are represented by the platform itself and the benefits it supplies to its users. Last but not least, the thesis covered a broad list of improvements that could be addressed within the scope of future work.

The core aspect of the present thesis is represented by the *OpenTravelMap* platform, a mobile location based service developed to assist users in creating and searching for travel routes. The developed platform also includes the connection with other platforms to offer direct access to external resources and a context aware recommender system that facilitates the process of decision making. The main target audience of the *OpenTravelMap* platform consists of tourists, which can access the features of the platform through their mobile devices.

The *OpenTravelMap* platform was developed as a client-server application able to run on Android mobile devices. The architecture of the platform comprises two major components: the *Create* and the *Search* component. The main purpose of the *Create* component is to provide tools that assist the user in creating travel routes. Each route consists of one or more locations created by the user. Moreover, the user has the possibility to edit the locations and save the entire route when desired. Through the *Search* component of the *OpenTravelMap* platform the sharing of information between users is enabled, since the data used by this component originates from the *Create* component. Beside the basic search functionality, the component also provides an advanced feature, a context aware recommender system. The recommender system operates by taking the user's preferences and relevant weather information into account from which it creates route recommendations that are further delivered to the user.

In conclusion, the current thesis submits a contribution to the field of tourism through the mobile location based service running on the *OpenTravelMap* platform. The outcome obtained within the present thesis encourages to further improvements that may be covered by future work.

Bibliography

[1] Android developers. https://developer.android.com/, accessed on May 1, 2018.

[2] bergfex tours & gps tracking running hiking bike. https://play.google.com/store/apps/details?id=com.bergfex.tour, accessed on December 23, 2018.

[3] Docker container running postgres with postgis. https://hub.docker.com/r/mdillon/postgis/, accessed on August 1, 2018.

[4] Flickr. https://www.flickr.com/, accessed on November 1, 2018.

[5] Flickr access photos. https://www.flickr.com/services/api/misc.urls.html, accessed on November 1, 2018.

[6] Flickr search photos. https://www.flickr.com/services/api/flickr.photos.search.html, accessed on November 1, 2018.

[7] Foursquare. https://foursquare.com/, accessed on November 1, 2018.

[8] Google play. https://play.google.com/store/apps, accessed on December 23, 2018.

[9] Map maker. https://play.google.com/store/apps/details?id=com.exlyo.mapmarker, accessed on December 23, 2018.

[10] Mapbox. https://www.mapbox.com/, accessed on May 1, 2018.

[11] Openweathermap. https://openweathermap.org/, accessed on November 1, 2018.

[12] Openweathermap 5 day weather forecast. https://openweathermap.org/forecast5, accessed on November 1, 2018.

[13] Openweathermap weather conditions. https://openweathermap.org/weather-conditions, accessed on November 1, 2018.

[14] Spring boot. http://spring.io/projects/spring-boot, accessed on August 1, 2018.

[15] Sygic travel maps offline & trip planner. https://play.google.com/store/apps/details?id=com.tripomatic, accessed on December 23, 2018.

[16] Travel map maker. `https://play.google.com/store/apps/details?id=ch.robera.android.travelmapmaker`, accessed on December 23, 2018.

[17] G. Adomavicius and A. Tuzhilin. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE Transactions on Knowledge & Data Engineering*, (6):734–749, 2005.

[18] L. Baltrunas, B. Ludwig, S. Peer, and F. Ricci. Context relevance assessment and exploitation in mobile recommender systems. *Personal and Ubiquitous Computing*, 16(5):507–526, 2012.

[19] P. Bereuter, R. Weibel, and D. Burghardt. Content zooming and exploration for mobile maps. In *Proceedings of the 15th AGILE international conference on geographic information science*, pages 74–80, 2012.

[20] J. Borràs, A. Moreno, and A. Valls. Intelligent tourism recommender systems: A survey. *Expert Systems with Applications*, 41(16):7370–7389, 2014.

[21] D. Burghardt, R. Purves, and A. Edwardes. Techniques for on the-fly generalisation of thematic point data using hierarchical data structures. In *Proceedings of the GIS Research UK 12th Annual Conference*, pages 28–30. Citeseer, 2004.

[22] J. Christopher B. and W. J. Mark. Map generalization in the web age. *International Journal of Geographical Information Science*, 19(8-9):859–870, 2005.

[23] K. Church, J. Neumann, M. Cherubini, and N. Oliver. The map trap?: an evaluation of map versus text-based interfaces for location-based mobile search services. In *Proceedings of the 19th international conference on World wide web*, pages 261–270. ACM, 2010.

[24] J. Gaffuri. Improving web mapping with generalization. *Cartographica: The International Journal for Geographic Information and Geovisualization*, 46(2):83–91, 2011.

[25] W. Gao, J. Gong, and Z. Li. Thematic knowledge for the generalization of land use data. *The Cartographic Journal*, 41(3):245–252, 2004.

[26] A. Kushwaha and V. Kushwaha. Location based services using android mobile operating system. *International Journal of Advances in Engineering & Technology*, 1(1):14, 2011.

[27] R. Looije, G. M. Te Brake, and M. A. Neerincx. Usability engineering for mobile maps. In *Proceedings of the 4th international conference on mobile technology, applications, and systems and the 1st international symposium on Computer human interaction in mobile technology*, pages 532–539. ACM, 2007.

[28] K. Meehan, T. Lunney, K. Curran, and A. McCaughey. Context-aware intelligent recommendation system for tourism. In *Pervasive Computing and Communications Workshops (PERCOM Workshops), 2013 IEEE International Conference on*, pages 328–331. IEEE, 2013.

[29] I. Muehlenhaus. *Web cartography: map design for interactive and mobile devices*. CRC Press, 2013.

[30] A.-M. Nivala and L. T. Sarjakoski. An approach to intelligent maps: Context awareness. In *The 2nd Workshop on'HCI in Mobile Guides*, 2003.

[31] B. Schmidt-Belz, A. Nick, S. Poslad, and A. Zipf. Personalized and location-based mobile tourism services. Workshop on Mobile Tourism Support Systems in conjunction with Mobile HCI, 2002.

[32] V. Setlur, C. Kuo, and P. Mikelsons. Towards designing better map interfaces for the mobile: experiences from example. In *Proceedings of the 1st International Conference and Exhibition on Computing for Geospatial Research & Application*, page 31. ACM, 2010.

[33] H. Yan and R. Weibel. An algorithm for point cluster generalization based on the voronoi diagram. *Computers & Geosciences*, 34(8):939–954, 2008.