Insert here your thesis' task.

Czech Technical University in Prague
Faculty of Information Technology
Department of software engineering

Master's thesis

"Coviator" - application for shared trip planning - OS Android

Georgiy Shur

Supervisor: Ing. Jiří Hunka

7th January 2016

Acknowledgements

First of all I would like to thank my supervisor Ing. Jiří Hunka for his patience and valuable advices. Many thanks to everyone on the IT faculty of CTU for sharing their knowledge. Moreover, thanks to all the people who helped me with prototype testing and support me during the writing of this thesis.

Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis.

I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular that the Czech Technical University in Prague has the right to conclude a license agreement on the utilization of this thesis as school work under the provisions of Article 60(1) of the Act.

In Prague on 7th January 2016 .....

Czech Technical University in Prague
Faculty of Information Technology

Citation of this thesis

Shur, Georgiy. *"Coviator" - application for shared trip planning - OS Android.* Master's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2016.

Abstrakt

Tato diplomová práce popisuje proces vývoje mobilní aplikace pro operační systém Android a serverové časti s databází, která komunikuje s mobilními klienty pomoci REST API. Výsledkem je funkční aplikace, která poskytuje uživatelům možnost plánování společného výletu s vlástními místy a navigaci mezi nimi. Tato práce se dotýká všech kroků softwarového vývoje: analýzy, návrhu, implementace a testování.

Klíčová slova Android OS, cestování, nativní aplikace

Abstract

This master's thesis describes the development process of a mobile application for Android OS and the server part with a database that communicates with mobile clients via REST API. The result is the functional app that allows its users to plan a shared trip with custom places and navigate through them. The work touches on all the steps of the software development: analysis, design, implementation and testing.

Keywords Android OS, travel, native app

Contents

List of Figures

List of Tables

Introduction

The idea of creating suchlike application occurred when I was traveling with my friends. We realized that even with the abundance of mobile and web applications presented in stores, it is quite difficult to find a tool that allows to plan a trip together from different devices, use this plan during travels even offline, and view and share its results in the end. But back then, it was only our humble opinion.

The culture of traveling changed a lot during last two decades due to technology advancement. By leaps and bounds Internet displaces television, radio and magazines as a travel information source, especially among young travelers. With so much information available, more and more people get used to organize their trips themselves rather than buying tours or packages from travel agencies. Everybody could get any information instantly, if they only had a smart device and Internet connection. GPS navigation allows smartphone owners to find their location and any places much easier than with paper maps. Transportation, accommodation, bars and restaurants, attractions, shops, excursions, events and just any random locations - all that can be found on the display of your smartphone or tablet. You don't need to use travel agencies anymore if you have a bit of time and savvy.

The other thing is, that people not only consume information, they produce it as well. Here comes the social networking. People like to share, and this statement is even stronger in the context of traveling. They share photos, news, locations, reviews. They share where they've been and what they'd like to visit. They are blogging about travels, discuss their experiences and give advices.

All these factors led to a rapid growth of the offer and demand of travel applications in mobile stores. There is a big variety of travel apps for different platforms and different purposes, but some of them are too specific, some have poor interface, some are overcomplicated, some, on the contrary, lack functionality. As long as each user has different needs, there is still a lot of opportunities in this field. This work will try to find these opportunities and

realize them in "Coviator", mobile application for shared trip planning.

This work consists of four chapters. First chapter is dedicated to analysis and research. In this chapter a little survey will be conducted, then we'll find out the main requirements for the application and analyze the existing alternatives. The architecture of application, communication protocols and user interface will be designed in the second chapter. The third chapter will include server and mobile client implementation details. In the last chapter, we'll try to test the app and apply modifications if needed.

Chapter 1

Research and analysis

1.1 Travelers research

Before we could come up with goals and requirements and choose platform for application, it's useful to conduct at least a little research among potential users. In our case the potential users are people who travel. First of all we are interested in travelers who own smart devices, but travelers without aforesaid could be our potential users too. It was decided that the most appropriate respondents for this research could be found in travel communities.

A little survey was published on several Czech, Russian and English speaking travel forums and social network groups. It was consisting of the following questions:

- How do you usually plan your trip (look for places to see and then navigate to them)? (open question)
- Do you use mobile applications for trip planning and navigation during your travels? (yes/no)
- What operation system does your smartphone/tablet use (single choice, response options: iOS, Android, Windows Phone, Other (name))
- What mobile applications do you use? (open question)
- What functions are important for you? (multiple choice, response options: Orientation on map, Points of interest (from guides or other users), Custom maps/places, Layers for different types of places, Shared access, Offline map availability, Feedback (photos, reviews, visited/unvisited), Social interaction (be a part of community), Route planning (by foot, car, public transport, etc.), Web version (for planning), Other (name))
- Do you miss something in your applications and would like to add/improve? (open question)

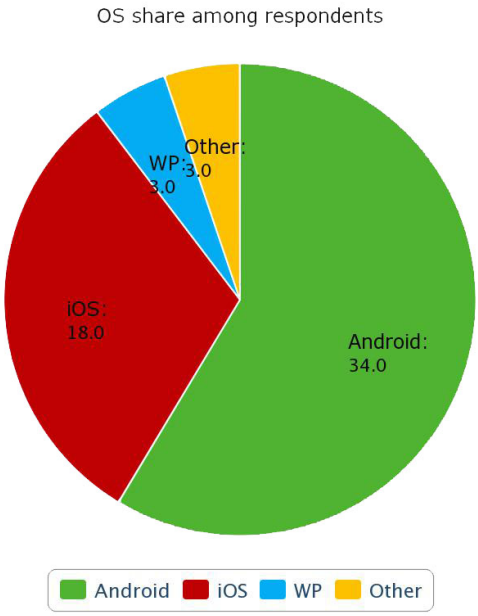


Figure 1.1: Mobile operation systems share among survey respondents

64 users were participating in this survey. Almost all respondents (91%) use smart devices for their trips. Most of them responded that they use different Internet resources and guides for trip planning or listen to their friends' or locals' advices. Then they're making a list of places of interest and use Google Maps or another maps, navigation systems or even screenshots in their travels. A few respondents who don't use smart devices answered that they're using printed maps.

The most popular operation system among respondents is Android (59%). They're also using iOS (31%), Windows Phone (5%) and some other platforms (5%)(figure 1.1)).

According to this survey, the most popular applications are Google Maps and Google Maps Engine. More then a half respondents use at least one of them. But some people aren't completely satisfied with it, because Google Maps application for all mobile platforms doesn't allow to create, edit, or view custom maps. They delegate those abilities to Google Maps Engine. But Google Maps Engine has limited functionality. For example, Android version doesn't support offline mode. Among other applications there were named Tripadvisor, Spotted by Locals, 2Gis, Maps 2Go, browser version of Google Maps, Navigon, OsmAnd, MAPS.ME and some others. The most popular applications that could compete with "Coviator" will be analyzed in detail in section 3 of chapter 1.

The next question of this survey attempted to list the main functions of

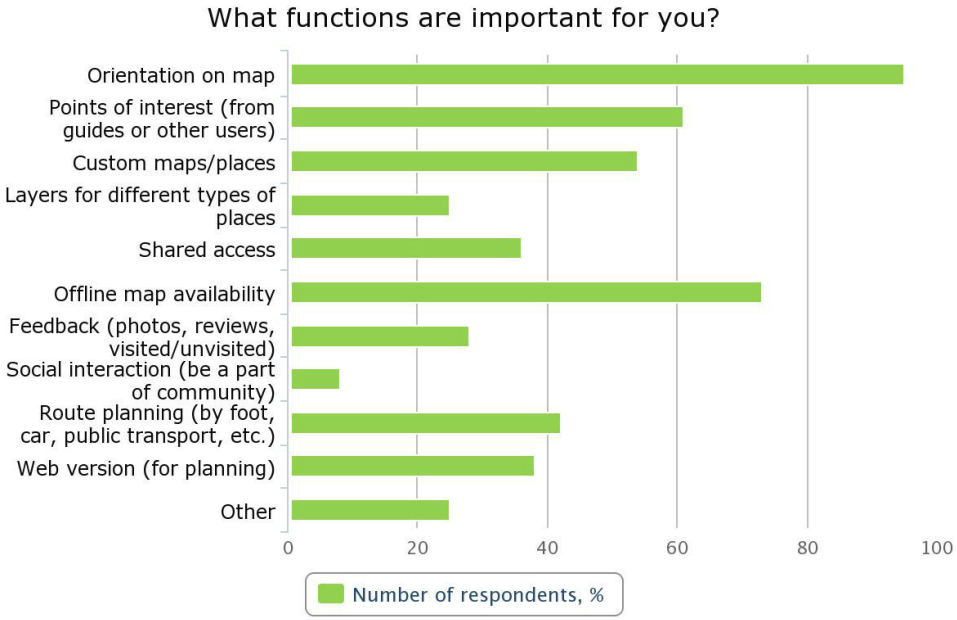


Figure 1.2: The most important travel application features among survey respondents

travel application. It was decided not to include flights and hotels booking functionality, because there are a lot of good applications and services with such specialization. Moreover, it isn't a good practice to mix such different things altogether, even if they belong to the same travels field. It might make the application overcomplicated and overflow it with needless functionality. The results could be seen on figure 1.2. Among other functions, the respondents named the opportunity to set different icons and colors do places on the map and the option to contact the place from the application interface. It's quite surprising that so few people are interested in social interaction. Maybe it's because people who plan their trips properly aren't interested in this aspect during planning phase and orientation. And then, they're used to share and discuss their experience in groups, social networks and forums known to them and depending on community they want to communicate with. Maybe that's why they don't want to see these features in their travel apps.

While answering the question what functions they're missing or would like to improve, a plenty of respondents named the functions from previous question, depending on what application they use. Moreover, some of them named friendly interface, speed, bike routes for cities and import from Google Maps.

The information obtained during this research will be the cornerstone for platform selection (chapter 1 section 2), goals and requirements specification

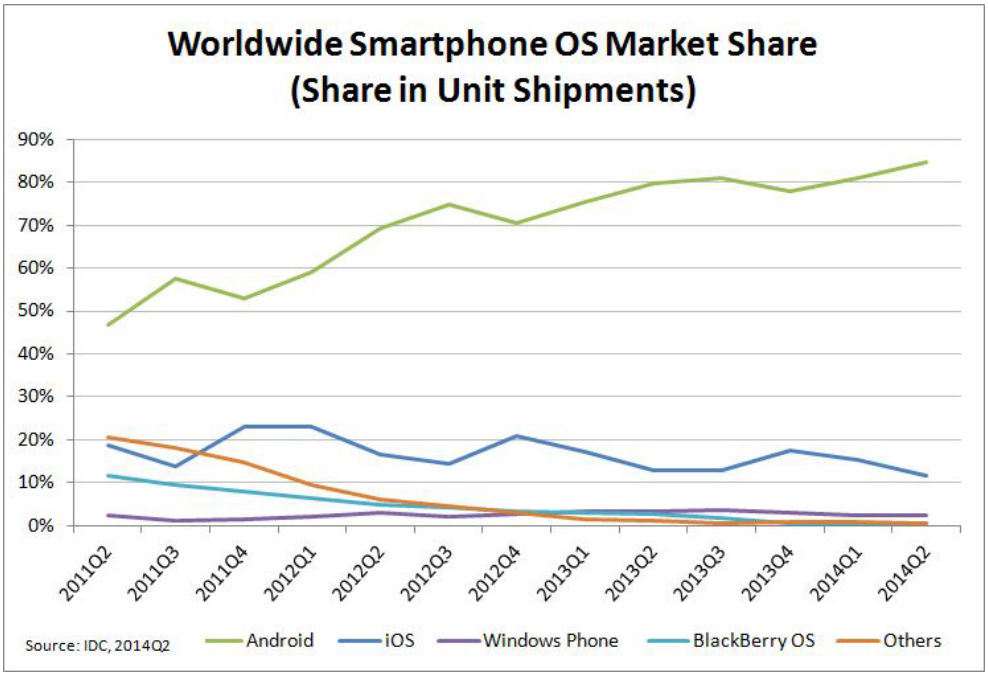


Figure 1.3: Mobile operation systems share worldwide

(chapter 1 section 4) and application improvement in the future.

1.2 Platform selection

The results of the survey from chapter 2.1 show that the most popular mobile platform is OS Android. But as far as the sample isn't representative, let's turn to some major statistics. According to IDC numbers for 2014 Q2[1], the situation is even more favorable for OS Android (fig 1.3)).

Anyway, iOS is still a very popular platform and according to our survey results, people are interested in web version as well. It's a good idea to make the application for multiple platforms as long as users may access the same functionality from different devices. This thesis' scope will be limited to OS Android only, but when developing communication API and server side, the possible application expansion on other platforms (web, iOS) will be always kept in mind. Moreover, in the conclusion part, we will consider this expansion as one of the ways the application may improve in the future.

Although OS Android is undoubtedly the most popular mobile operating system, there is a lot of obstacles in application development for this platform.

The first obstacle is the wide range of OS Android versions that are still in use. Some new functions, that are available for newer versions, aren't supported by the older ones. On the contrary, the methods used for the

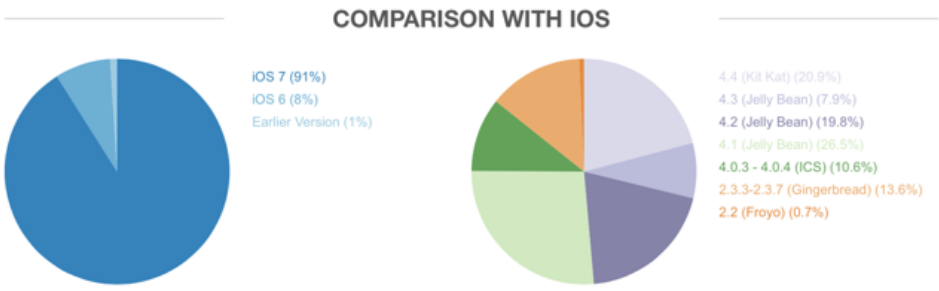


Figure 1.4: Comparison of iOS and OS Android versions share for August 2014, AppleInsider

development of older versions rapidly become deprecated. In other words there is a big problem of both forward and backward compatibility. The other problem is the open-source nature of Android. Even the same version may look different on different manufacturers' software suites. This software fragmentation may become android developer's headache. The solution is to analyze, what versions are the most popular and develop with the latest widely adopted version in mind. On the figure 1.4 you can see the actual market share for iOS an OS Android different versions[2].

In November 2014, the new version of Android will be released - 5.0 "Lollipop". And while the application will be in development, the share of the older version 2.3 "Gingerbread" will only decrease. So, it was decided to support versions 4.0.3 "Ice Cream Sandwich" (API level 15) and higher and choose version 5.0 "Lollipop" as the target version.

The second big problem is hardware fragmentation. There is a vast variety of different devices running OS Android (more than 12000), that have different shapes, screen sizes, keyboards, cameras and buttons. It's almost impossible to develop an application that will work as expected on all Android devices. The solution is to choose the most popular devices from each size category and manufacturer and test the application on them. When it will look and work as expected, we can eventually expand it on other devices.

1.3 Existing alternatives

Now, knowing the platform for our future application and its rough concept, we could monitor the situation on the market and analyze the possible competing products in detail.

1.3.1 Maps (Google inc.)

Author: Google inc.

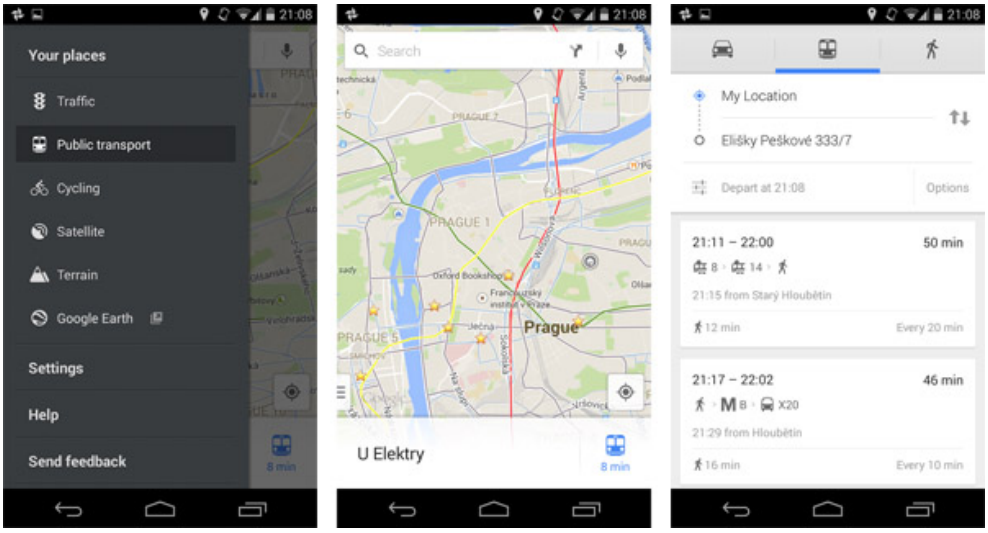


Figure 1.5: The screenshots of Google Maps

Updated: October 24, 2014

Downloads: more than 1 billion

Price: free

It sounds funny when we mention Google Maps among competing applications. It's impossible to compete with Google Maps, because almost every Android user use this application. But when we look closer, we'll see, that Maps hasn't even a half functionality we would like to achieve. The old browser version of Google maps had a lot of that stuff: custom maps, layers, different colors and icons for places, shared access, KML import, etc. But then they decided to split functionality between two applications: Maps itself and so called Google Maps Engine (GME). All the features related to custom maps were delegated to GME, and the only way you can save places in Maps now - add them to "Favourites" without any classification options. But still, Maps is a great app for navigating, it has a huge base of places, routes and other information the most popular maps application should have.

The other thing is that Google maps has great open API that allows any developer to use a wide amount of features and add their own. It is the most popular maps API, that is available for iOS, OS Android and web. We will consider the opportunity of using Maps API for our application in the next chapter.

Regarding the facts above, we won't put Google Maps in the resulting table of alternatives.

1.3.2 My Maps (former Google Maps Engine)

Author: Google inc.

Updated: September 24, 2014

Downloads: more than 1 million

Price: free

Google has made a lot of changes in its Maps product during last two years. Along with the significant changes in application user interface for all platforms, Google picked out some of the functionality in separate applications. Thus Google Maps Engine appeared. Even if this idea of encapsulate all custom maps functionality in separate application and not to overload original Maps sounds good, the realisation for OS Android is quite controversial. For example, you can't access to Maps "Favorites" from GME. Or if you want to use navigation, you still need to switch to Maps. So, instead of separation we can see duplication here. The reviews in Play Store are showing that the users aren't satisfied as well. But Google is constantly improving its products, and it's just a matter of time when the changes occur. Anyway, analysis of Google inc. is beyond this thesis' scope, let's take a look at the My Maps application for OS Android.

The My Maps functionality isn't very rich, but it contains the main features for custom maps management. It is based on Google Maps and some functionality is preserved here: access to Maps points of interest, search, navigation, satellite view. Moreover, the user can sign in with its Google account and immediately gain access to the custom maps associated to this account. My Maps can create and edit custom maps, split it to different layers, add and edit custom places to layers(either from coordinates or Google base or by touching the map), add description to them and switch to Maps for routing. We can access shared maps here, but can't share it. The big advantage of this application that it exists also for other major platforms: web and iOS. Web version is more functional and polished. Here user can share maps, create not only custom places, but custom routes too, export KML files and some other things. Basically, the mobile version of My Maps could be seen as some addition, viewer for the main, web application. But the mobile version gets new functions time by time. One year ago, it could only view custom maps and nothing else.

Now, the shortcomings of My Maps for OS Android. The first one - the app doesn't support offline mode. You could neither download map regions nor see the labels. And it's quite strange, because iOS version works offline and offers to user to download a region on map. The second flaw is its speed. Because with every zoom, the app downloads every chunk of map with labels as well, it lasts a couple of seconds before we can see the full image. Our respondents mentioned it as well. The iOS version doesn't have this problem.

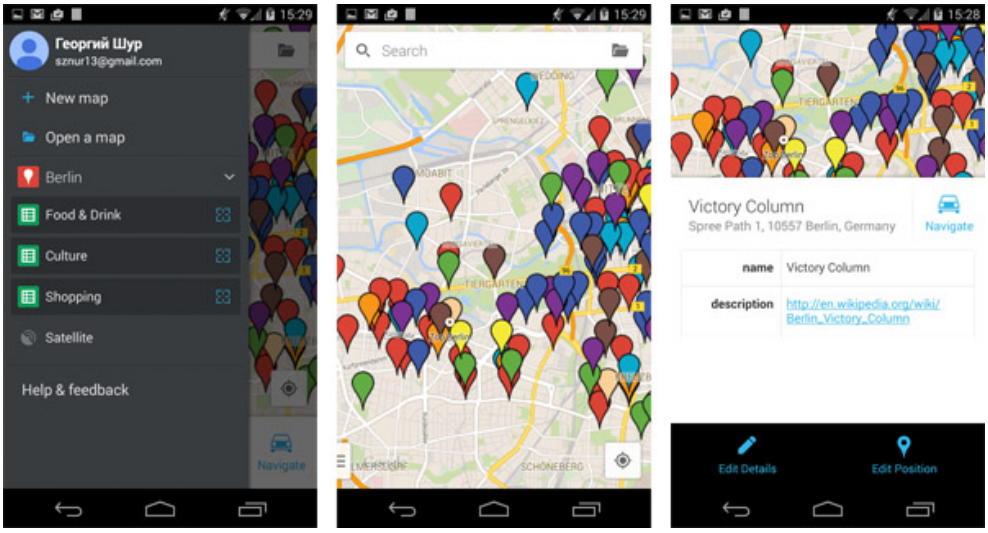


Figure 1.6: The screenshots of My Maps

The third one - users can see custom places only on the map, the app doesn't have list view. It could be a problem when you have more than hundred of places. And in the end, it isn't entirely a travel app. It hasn't the whole feeling of a trip. You can't add the feedback, mark visited places, summarize the results and share them with other people.

Of course, My Maps is one of the main applications with such functionality, and the most popular. It has some drawbacks, but we'll try to fill these lacks and to take the best from it.

1.3.3 MAPS.ME and MAPS.ME Pro

Author: MapsWithMe GmbH

Updated: October 8, 2014

Downloads: more than 1 million and more than 100 000 for pro version

Price: free and 4.99$ for Pro version

It's quite hard to find good maps application for Android, that don't use Google engine. But there are some really good maps that even overcome it in some characteristics. MAPS.ME is one of them. Fast, detailed, looking good, completely offline and has an impressive database (from OpenStreetMap data) of points of interest (POI). Moreover, the search works in offline mode (only Pro version) and supports classification and filtering. All you need is to load the country or city of interest in advance. Maps aren't big and have different level of detailing depending on zoom.

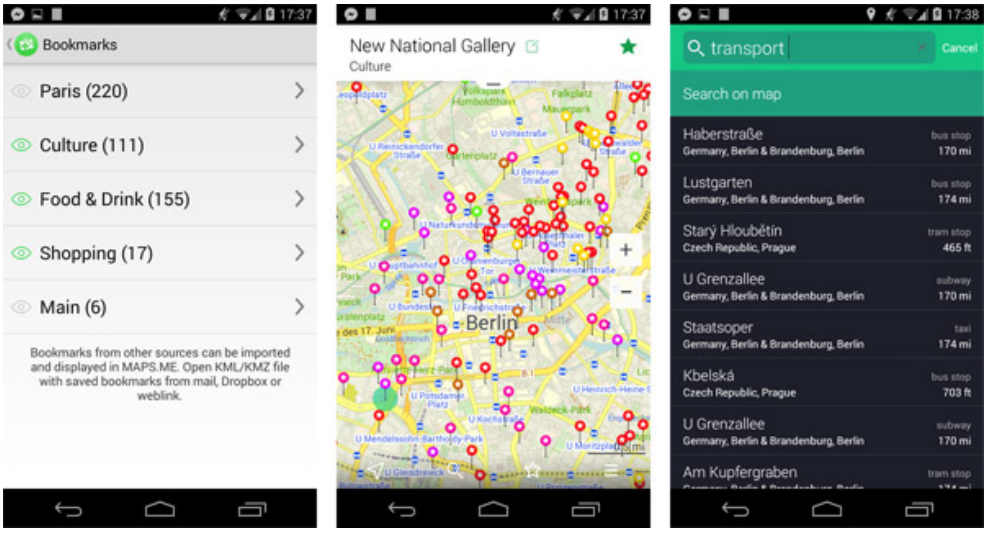


Figure 1.7: The screenshots of MAPS.ME

MAPS.ME Pro allows to make your own bookmarks (places) from coordinates, database or by touching the map and add them to custom sets (layers). Also you can edit names, descriptions and colors (only 8). These bookmarks are available offline. The app supports GPS positioning routing in offline mode. The public transport information isn't complete. MAPS.ME supports KML import. The application doesn't have web version, but it exists for other mobile platforms.

Furthermore, you can download guides for free if you have Pro version. They are separate applications and are available for more than 20 countries. Still, it is a map application, not consistent travel app.

1.3.4 TripAdvisor

Author: TripAdvisor

Updated: October 17, 2014

Downloads: more than 100 million

Price: free

Another one very popular service for travelers, that has some common functionality with our future app, but won't be a directly competing application. TripAdvisor has its own base of restaurants, hotels, attractions, tours etc. with detailed information and massive feedback potential. TripAdvisor is used commonly by users to find places or activities according to ratings, based on user reviews.

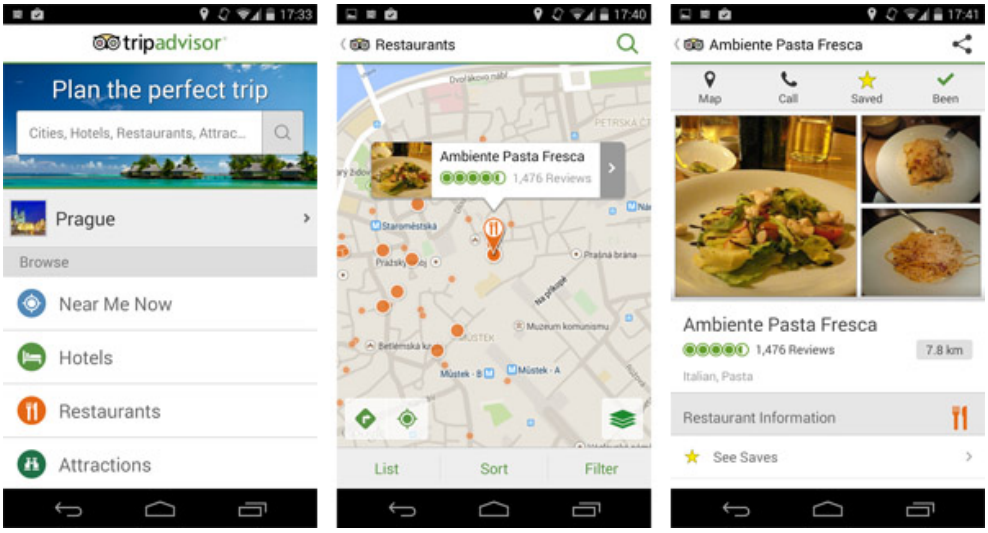


Figure 1.8: The screenshots of TripAdvisor

TripAdvisor for OS Android has two kinds of presentation: map and list. It provides filtering and location-based search. Then, you can add places to "Favorites" for each city, but only places from their database. TripAdvisor doesn't have shared access function and doesn't work offline. But you can download additional guide applications for each city with the same functionality, but working offline. TripAdvisor has web and iOS version. You can access saved places from all platforms on account basis.

TripAdvisor allows users to mark visited places, leave reviews, photos and ratings that will be available for other users. Some social interaction functions are avilable as well.

1.3.5 Tripomatic

Author: Tripomatic

Updated: October 6, 2014

Downloads: more than 100 000

Price: free, 5.49$ for offline maps (in-app purchase)

The first application from this list that is developed for complete itinerary planning. The app exists for Android, iOS and web. Tripomatic allows to create your trip from the beginning, then add places to each day of the trip. It has its own base of places filtered by type, but allows to create custom places and add category tags and description. The app allows to view your itinerary on the map or in the form of daily list or overall list. Tripomatic supports offline maps of top destinations only for additional price. Users may share their

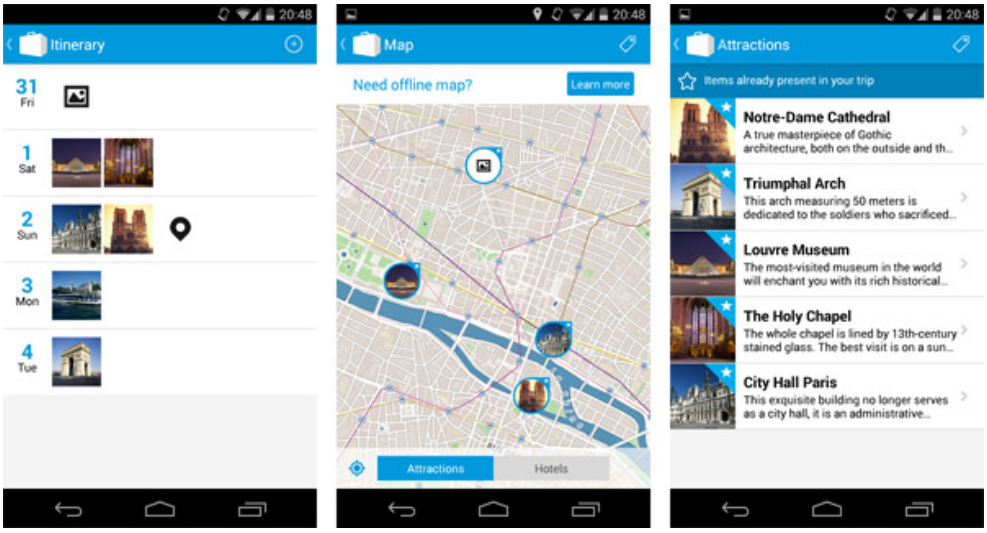


Figure 1.9: The screenshots of Tripomatic

trips, but not for editing, only for observation. The routing option is present, but uses Google Maps app (web version uses its own routing). Moreover, this application can search for tours, tickets, car rentals and forecast weather. Among other advantages we could name the good look of the application.

Unfortunately, upon close examination, the app appears to be raw. For example, you can see custom places on the map and in the daily list, but they are missing in the places overall list. Even if the app has days separation, you can't filter places on the map by days or open a map for particular day (web version doesn't have this drawback). The offline map supports only base places, not user mapped itineraries. The places are displayed in the lists by pictures, but you can't add pictures to custom places. Then, the routing isn't available for custom places.

This is a good application with wide functionality, but many little drawbacks and lack of shared access make it far from ideal, and controversial reviews on Play Store confirm it.

1.3.6 Roadtrippers

Author: Roadtrippers

Updated: October 30, 2014

Downloads: more than 100 000

Price: free

Another application that allows to create complete trip itineraries. Web, Android and iOS versions are available and look good. Unlike the previous

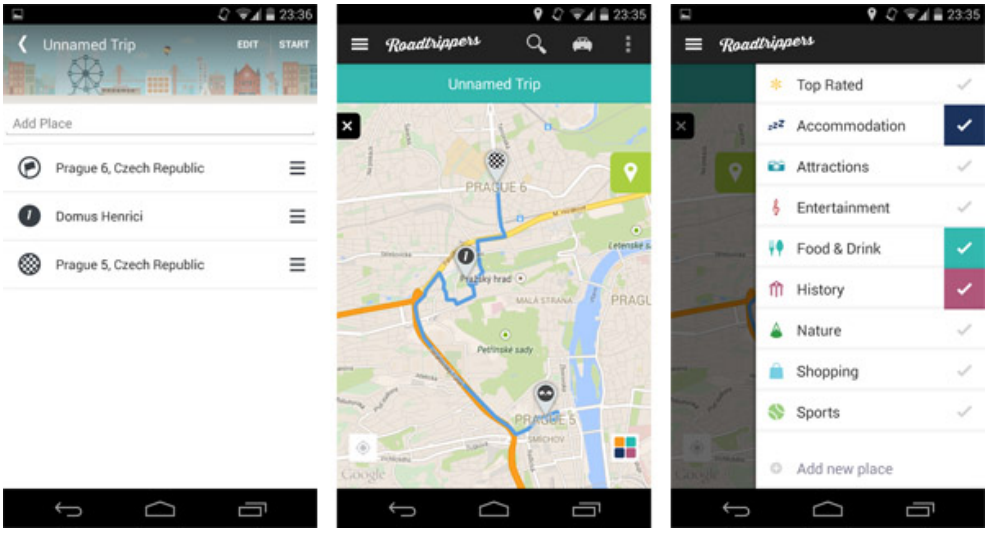


Figure 1.10: The screenshots of Roadtrippers

app, Roadtrippers has very friendly user interface. Everything is clear and works as expected. Unfortunately, this application supports fully only USA at the moment. But you can already find some places in Europe and authors promise to support other countries in the closest future.

The trip planning begins by entering the start and finish location. Then the app suggests the places for this route and allows to add them. Moreover, you can see places independently of your route, filter them by type and subtype and add to the custom collections. Roadtrippers has big database of places and each user may leave ratings, reviews, photos etc. Then, you could add your own place (leave pin on the map or write an address) and it will be immediately added to their database. So you can't add custom places only for yourself. The app calculates expected distance, time and gas expense for your route.

Unfortunately, Roadtrippers for Android doesn't support offline mode. Moreover, it seems to make some complicated calculations and sometimes works slow. For example, if you'll try to add some African place to your trip from Prague 5 to Prague 7, the app will freeze with the only opportunity to restart it. The trips can be shared for observation, but not for editing.

1.3.7 Summary

Of course, there are much more trip planning applications on the market, but they are either less successful copies of the apps mentioned above or aren't enough usable or popular.

The main functions and features of analyzed apps are summarized in the table 1.1). "+-" means that the feature is supported partly, for additional

cost or uses third-party application.

Features	MyMaps	MAPS.ME	TripAdvisor	Tripomatic	Roadtrippers
GPS position	+	+	+	+	+
Places search	+	+	+	+	+
Guides	-	-	+	+	+
Places saving	+	+	+	+	+
Custom places	+	+	-	+	+-
Layers/filters	+-	+	+-	+	+
Shared editing	+	-	-	-	-
Offline mode	-	+	+-	+-	-
Feedback	-	-	+	+-	+
Routing	+-	+	+-	-	+
Web version	+	-	+	+	+
KML import	+-	+	-	-	-

Table 1.1: Applications' features comparison

As we can see, each of these applications lacks some features. Besides, some of them have problems with speed, UI or reliability. This work will try to take these flaws into account and take only the best from these apps.

1.4 Requirements specification

Now, with the information obtained from survey and from alternatives analysis, we should be able to formulate functional and non-functional requirements.

1.4.1 Functional requirements

FR1 Account creation and management Since it is an application with shared access that will be used by friends, creating an account will be obligatory. Complicated authentication system is unnecessary, but some simple and fast sign up/sign in mechanism is a requirement. During registration, the user will be prompted to enter his screen name, valid email address and password. Then, he will be able to sign in with email/password pair. In view of cases when several users may use the same device, the sign out option will be available as well.

FR2 Trips displaying The user should be able to easily look through his trips and distinguish between different types if there will be some.

FR3 Trips creation, updating an deleting Each user should be able to create his own trip and optionally add other travelers (coviators) to it. Some additional options like attaching a picture should be available.

FR4 Trip places listing When opening trip detail, user is expecting to see its places in suitable form like scrollable list.

FR5 Trip places map One of the main functions of this application is navigation on map. Trip places should be displayed on map in the form of markers. For better user experience, it's good to add such useful and familiar functions like positioning and navigation.

FR6 Places creation, updating an deleting Similar to trips but with some additional functions. It is good idea to offer the users to choose place location on map manually, or through familiar to him Google Maps interface with places search. To allow users to quickly browse through places, they could be optionally classified to user-created categories/layers with their own colors. There will be the possibility to add some description to place (address, history or some other notes). The user should be able to mark the place visited as well.

FR7 Place detail The detailed information about particular place that isn't presented in the list, should be easily accessible from both the map and the list.

FR8 Categories creation, updating an deleting Categories shouldn't be complicated. Just name and color should suffice. Color then will be used in both the list and the map.

FR9 Place filtering and search Filters are useful when the user is interested only in some limited categories of places. The application should allow filtering at least by user-created categories and by "visited" mark. Sometimes the trip may contain tons of places, and scrolling through all of them may be annoying, that's when search by name is irreplaceable, especially real-time search.

FR10 Info and contacts Some contact on administrator/developer/operator should always persist in the application. It may be the electronic form or just static screen with information. A reference on info or help page won't be redundant as well.

1.4.2 Non-functional requirements

Client-server architecture Application will have client and server sides. Server side will store personal and shared data (accounts, trips, contributors) and serve as a communication medium between end devices.

Clients will store data available for offline use and display it on user request.

RESTful API Communication between end devices and server will be performed through HTTP-based REST API. The information will be exchanged in JSON format. Such architecture enables easy further expansion on other platforms.

Android OS platform Client side will be developed only for OS Android due to reasons discussed in chapter 1.2. In the future, it may be extended to iOS and web platforms. Minimum supported API level of Android SDK will be 15 (version 4.0.3 "Ice Cream Sandwich"), target API level will be 21 (version 5.0 "Lollipop").

Shared access Multiple users should be able to view or/and edit (depending on access level) data from the same trip. The data about trip/place creator will be available on server and displayed in application.

Offline availability The major part of functionality should be available without Internet connection: map, trips, places displaying, filtering and navigating on map. Creation, edition and deletion will available only with Internet connection. Such limitation allows to avoid different synchronization problems.

Internet connection for server actualization Without connection, end devices will have access only to local copy of data. But for obtain actual data from the server or apply changes to shared trip, they should be connected to the Internet. Each time the data will be received from server, it will be synchronized with local database.

Google Maps mapping API The application will use Google Maps API because it's the most popular mapping API for Android. It provides Google maps functionality and allows to overlay it with your own. It isn't complicated and its interface is quite familiar to Android users.

Android design guidelines compliance To simplify users' experience with the application it's a good practice to follow these principles when developing for Android.

Stability and quickness The application should avoid crashes, freezes and long pauses during some tasks (for example zooming or filtering).

Simplicity of use The application should have friendly interface that average user could quickly understand by himself.

Hints and feedback The application should always inform user about what's going on. It includes touch feedback, pop up messages (Toasts and

Snackbars in Android), progress bars, input error messages, confirmation dialogs.

Chapter 2

Design

2.1 Use cases and scenarios

According to Ivar Jakobson, the inventor of use case concept, a use case is all the ways of using a system to achieve a particular goal for a particular user. Taken together the set of all the use cases gives you all of the useful ways to use the system, and illustrates the value that it will provide[3].

More specifically, a use case is a written description of how users will perform tasks on your website. It outlines, from a users point of view, a systems behavior as it responds to a request. Each use case is represented as a sequence of simple steps, beginning with a user's goal and ending when that goal is fulfilled[4].

Use cases are typically the next, more detailed level of functional requirements. Use cases are derived from functional requirements, but not necessary have one-to-one correlation. In many situations, each functional requirement will break down into several use cases. Use cases list may be graphically visualized with UML use case diagram.

For better understanding and clearer outlook on the scope of the project, use cases are often accompanied with textual descriptions - scenarios. A scenario is one path or flow through a use case. Typically a use case has a primary scenario, one or more alternate scenarios, and possibly exception scenarios[5]. Now, let's try to work out the use cases based on functional requirements and describe their flow with scenarios.

UC1 Sign in Signing in is a required action for user to use the application. Sign in will be available only if the user isn't already logged in. Login credentials should be persisted on the device for the user not to be forced to enter the data each time he opens the app.

1. The application will display the login screen with data form.
2. The user will enter the login credentials in the form and send them.

3. The application will verify this data. If it is valid, the user will be logged in, else, the error message with the cause will be shown.

UC2 Register If the user doesn't have an account yet, he is allowed to create one through application interface. Registration is still available only if the user isn't signed in.

1. The application will display the registration screen with data form.
2. The user will enter the desired credentials in the form and send them.
3. The application will verify this data. If it is valid, the user account will be created and the user will be automatically logged in, otherwise, the error message with the cause will be shown.

UC3 Sign out The user will be able to sign out from the application. Sign out is available only if the user is signed in.

1. The user chooses the "sign out" option.
2. The confirmation dialog is shown to him.
3. If the user confirms logout, the application goes to the login screen. Else, the user stays inside the main application part logged in.

UC4 Display trips The list with user trips will be shown to the user. The trips screen is supposed to be the home screen when the user is signed in.

1. The user chooses the "my trips" option.
2. The application displays the trips list.

UC5 Create trip The user could create his own trip.

1. The user chooses the "create trip" option.
2. The application shows the create trip form.
3. The user fills the form with data and sends it to the application.
4. If the data is valid, the application creates the trip and adds it to the trip list. This trips becomes available for other users added to the trip by the creator. Otherwise, the error message is shown to the user.

UC6 Edit trip The user could edit his existing trip.

1. The user chooses to edit the trip.
2. The application shows the edit trip form (same as create with pre-filled trip data).
3. The user may edit or add new data to the form and send it.

4. If the data is valid, the application edits the trip. Otherwise, the error message is shown to the user.

UC7 Add/remove coviator When creating or editing his trip, the user is able to invite other people (coviators) to his trip or kick them out. The added coviators should be displayed on the screen for user to be able to remove them.

1. The user chooses to add coviator to the trip.
2. The application prompts the user to find an existing coviator and add him to the trip.
3. If the coviator exist and isn't already in the trip, the application adds him to the trip and show to the user. Otherwise, the error is shown.

UC8 Delete trip The user will be able to delete his own trips.

1. The user chooses to delete trip.
2. The confirmation dialog is shown to him.
3. If the user confirms deleting, the application erases the trip and all places and categories inside it.

UC9 Display places list The user will be able to browse through places inside his trips.

1. The user chooses the trip.
2. The application shows the places belonging to this trip.

UC10 Filter places The user should be able to filter his places list by different parameters.

1. The user chooses the "filter" option.
2. The application shows to him the different filter parameters.
3. The user chooses the desired filter set and sends it to the application.
4. The application shows the user the places list filtered by the user request.

UC11 Search places The user should be able to search places inside trip by name.

1. The user chooses the "search" option.
2. The application allows the user to type the name of the place.
3. The user enters the name of the place.
4. As the user enters the name to the field, the application shows in real-time the places that are corresponding to the entered query.

UC12 Display places map The application will display trip places on the map in the form of markers. The markers should be located according to the places coordinates and be distinguishable by their category. The filters and search results from the list should work on the map markers as well.

1. The user chooses the "show on map" option.
2. The application shows the places belonging to this trip on the map.

UC13 Go to my location The orientation on the map is one of the main functions of the application. The user should be able to quickly find his own location on the map.

1. The user chooses "go to my location" option.
2. If positioning is available on the device, the application moves the map viewport to the coordinates of user current location.

UC14 Navigate to place The user should be able to navigate to the chosen place.

1. The user chooses "navigate" option.
2. The application shows the navigation directions to the user.

UC15 Show place detail The application will be able to show the user detailed place information that isn't presented in the list.

1. The user chooses the place.
2. The application shows detailed information about the place.

UC16 Create place The user should be able to create places that will be then added to particular trip. The created place will be immediately available to other coviators.

1. The user chooses the "create place" option.
2. The application shows the create place form.
3. The user fills the form with and sends it to the application.
4. If the data is valid, the application creates the place inside the chosen trip and adds it to the places list. Otherwise, the error message is shown to the user.

UC17 Edit place The user should be able to edit an existing place in his trip.

1. The user chooses the "edit place" option.
2. The application shows the edit place form with pre-filled place data.

3. The user may update the data and send it to the application.
4. If the data is valid, the application updates the place data. Otherwise, the error message is shown to the user.

UC18 Choose location on map When creating or updating the place, the user should be able to choose location for his place. There are two possible scenarios.

First scenario:

1. The user chooses the "choose place location on map" option.
2. The application shows displays the map with search bar.
3. The user enters the query to the search bar (name, address, coordinates etc.).
4. The applications suggests the user the predictions and finds the location according to user query.
5. The user chooses this location and returns from the map or continues the search.

Second scenario:

1. Same as UC18.1 first step.
2. Same as UC18.1 second step.
3. The user chooses the location directly on the map without entering any text data.
4. The application computes coordinates of this place and offers the user to save it.
5. Same as UC18.1 fifth step.

UC19 Choose category During place creation or edition, the user should be able to choose the category from the list and apply it to this place.

1. The user chooses the "choose category" option.
2. The application shows the list of existing categories and the option to add new one.
3. The user chooses the category.
4. The application saves user choice for the place.

UC20 Delete place The user should be able to delete places inside his trip.

1. The user chooses to delete place.
2. The confirmation dialog is shown to him.

3. If the user confirms deleting, the application erases the place.

UC21 Display categories The application will show the list of existing categories for particular trip.

1. The user chooses to edit categories or chooses category for place.
2. The application displays categories list.

UC22 Create category The user should be able to create custom category for particular trip.

1. The user chooses the "create category" option.
2. The application shows the creation form.
3. The user fills the form with and sends it to the application.
4. If the data is valid, the application creates the category for the cosen trip and adds it to the categories list. Otherwise, the error message is shown to the user.

UC23 Edit category The user should be able to edit categories.

1. The user chooses the "edit category" option.
2. The application shows the editing form with pre-filled category data.
3. The user may update the data and send it to the application.
4. If the data is valid, the application updates the category data. Otherwise, the error message is shown to the user.

UC24 Delete category The user should be able to edit categories.

1. The user chooses to delete category.
2. The confirmation dialog is shown to him.
3. If the user confirms deleting, the application erases the category.

UC25 Display app info The application should show user some basic information about itself and developer/operator contacts.

1. The user chooses the "app info" option.
2. The application displays the info.

The main use case diagram is shown in the figure B.1.

The use cases are describing the system from the user point of view, when functional requirements - from the system itself. Even when these approaches are containing different levels of granularity, they're about the same system, hence, they should cover each other completely. The table 2.1 depicts how our use cases cover the functional requirements given beforehand.

	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10
U1	■									
U2	■									
U3	■									
U4		■								
U5			■							
U6			■							
U7			■							
U8			■							
U9				■						
U10									■	
U11									■	
U12					■					
U13					■					
U14					■					
U15							■			
U16						■				
U17						■				
U18						■				
U19						■				
U20						■				
U21						■		■		
U22								■		
U23								■		
U24								■		
U25										■

Table 2.1: Functional requirements covered by use cases

2.2 Domain model

The use cases and scenarios define the behavioral aspects of the system, but they don't describe the structure. Sometimes the the transition between behavioral model and more specific structural (database or classes) model can be quite hard. For these purposes exists the domain model, that models the structure, but without implementation-dependent details.

Domain Modeling is a way to describe and model real world entities and the relationships between them, which collectively describe the problem domain space. Derived from an understanding of system-level requirements, identifying domain entities and their relationships provides an effective basis for understanding and helps practitioners design systems for maintainability, testability, and incremental development. Because there is often a gap between understanding the problem domain and the interpretation of require-

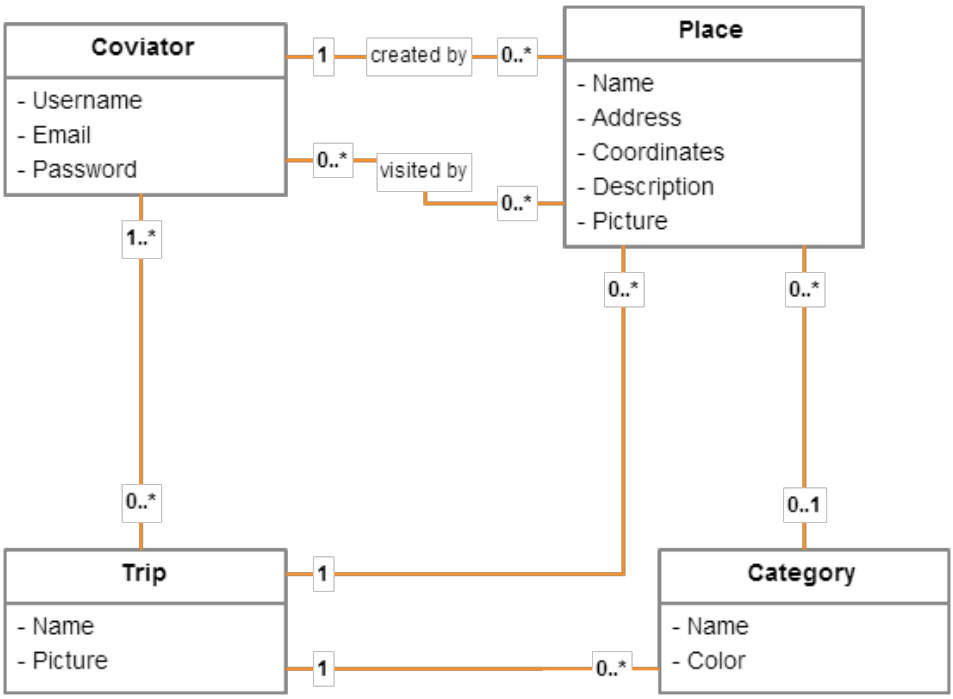


Figure 2.1: Domain model

ments, domain modeling is a primary modeling area in Agile development at scale. Driven in part from object-oriented design approaches, domain modeling envisions the solution as a set of domain objects that collaborate to fulfill system-level scenarios[6].

The domain model for Coviator is depicted in the figure 2.1. Let's describe the entities presented there.

2.2.1 Coviator

This is the user entity. It will contain the information about all the application users. Each registration will add one user object to the system. Data like email and password should have limited access, some security mechanisms, and must not be exposed from the server (in detail in further chapters). Each coviator could have any number of trips (create or be invited).

2.2.2 Trip

This is the main application entity. When trip is created by user, any amount of places, categories or coviators could be added to it. Coviators could attend any number of trips, but each place or category may belong to one trip at most. It will be possible to attach a picture to the trip.

2.2.3 Place

This is the entity that will keep all the information about places (points of interest) of the trip. The place must have its name and coordinates for displaying on the map. Moreover, it should be possible to add some description to the place, attach a picture and assign a category. A place may be attached to one category at most. The place should contain the information about who is the creator and who did visit this place among the coviators. The place should have only one creator and any number of visitors.

2.2.4 Category

The category is a simple entity that have only two attributes: name and color, both are required. The color is used to distinguish the places of particular category on both the list and the map. The category may belong to one trip only and any number of places could be assigned to it.

2.3 Backend

Since the application should provide shared access for multiple users around the world through Internet, the backend solution have to be designed as well. The client-server architecture was chosen as the most suitable for us for the reasons of simplicity, centralization, data persistence and accessibility.

The main server purposes will be receiving data from clients, storing data in the database and sending this data back to clients. Due to our requirement that application should work locally even in offline mode, the client shouldn't take data directly from the server. The server should rather provide synchronization for end devices when they're requesting it. Hence, we don't have great requirements on server load and excessive performance. The server should also provide basic authentication functionality.

Considering our server isn't much sensitive to performance, it was decided to take maintainability, simplicity and platform independence as priorities in technology choice. Besides, the programming language preferences of the author were taken into account. For these reasons it was decided to build the server with some of Java frameworks. There were two options considered: Spring MVC and Play!. Below are the advantages of each of analysed frameworks.

Spring MVC:

- Better history
- Better documentation
- Bigger community
- Broader functionality

Play!:

- Shorter learning curve
- Better performance
- RESTful (made by Web developers)
- Better out-if-the-box functionality
- Faster testing
- Simplier to use

The choice was quite obvious, especially when Play! doesn't lack any important functionality that Spring has. The main Play! language is Scala, but it supports Java as well.

For database purpose we will use some ORM library. Since ORM isn't good choice for Android client due to hardware and performance issues related to reflection use, for server it's just ok. ORM libraries allow to map database entities directly to objects, POJOs (plain old Java object) in our example. Play! framework supports two ORM libraries: JPA and Ebean. After a little study, it came out that for our purposes, there aren't much differences between them. But due to better documentation and some extra features, Ebean was chosen.

For communication between server and client it was decided to use RESTful API over HTTP protocol. It's simple, uniform, popular and multiplatform. Moreover, Play! framework has a great support of RESTful services, so it simplifies the development vastly. As the data format, our API will use JSON, the first class citizen both in Android/Java and Play! framework.

According to our use cases, the list of HTTP requests was created. [7] was used partly as an API design guideline.

POST /register Registers new user. Credentials should be sent in JSON. Returns 201(created) with the username in case of success or 400(bad request) with the details if credentials are wrong or user/email already exists.

GET /login Serves for the verification of user credentials and allows him to access application data. It's always good practice to provide at least minimal security to authentication even if your application doesn't store any valuable data. In our example it was decided to confine with Basic Authentication: the encoded string is sent by the client in the "Authorization" header of request and then decoded on the server side. But in most cases it isn't sufficient. Moreover, it is useless without SSL (HTTPS protocol), because it provides just basic BASE64 encoding that can be easily decoded [8].
Returns 200(ok) if succeeded or 401(unauthorized) with the details if the password is invalid or the user doesn't exist.

GET /users/autocomplete Requests existing usernames starting with queried char sequence. Returns 200(ok) with user list if succeeded.

GET /trips Requests trips for the queried username. Returns 200(ok) with trips list if succeeded or 404(not found) if user doesn't exist.

POST /trips Creates new trip. Accepts JSON with trip data. Returns 201(created) with the created trip object on success.

PUT /trips/{id} Updates existing trip. Accepts JSON with trip data. Returns 200(ok) with the updated trip object on success.

DELETE /trips/{id} Deletes existing trip. Returns 204(no content) if deleted successfully.

GET /trips/{id} Requests trip detail for requested ID. Returns 200(ok) with requested trip object on success.

GET /trips/{id}/users Requests users for particular trip. Returns 200(ok) with users list if succeeded.

POST /trips/{id}/places Creates new place for particular trip. Accepts JSON with place data. Returns 201(created) with the created place object on success.

DELETE /places/{id} Deletes existing place. Returns 204(no content) if deleted successfully.

PUT /places/{id} Updates existing place. Accepts JSON with place data. Returns 200(ok) with the updated place object on success.

PUT /places/{id}/visit Changes the "visited" state of particular place by particular user. User authorization is requested in the header. Returns 200(ok) if the state was changed successfully.

GET /trips/{id}/categories Requests categories for particular trip. Returns 200(ok) with the list of categories for requested trip.

POST /trips/{id}/categories Creates new category for particular trip. Accepts JSON with category data. Returns 201(created) with the created category object on success.

DELETE /categories/{id} Deletes existing category. Returns 204(no content) if deleted successfully.

PUT /categories/{id} Updates existing category. Accepts JSON with category data. Returns 200(ok) with the updated category object on success.

2.4 Database model

Now, with the requirements specified, use cases described, technologies chosen and domain model ready, we could proceed with more specific structural design. It have to be defined how all the will be persisted. Because of slightly different requirements, database models of client and server parts will differ too.

2.4.1 Server

Let's come up with the brief description of our database tables. The detailed model is presented on the figure 2.2.

Coviator The first and the main difference between server and client database model is the responsibility of the latter to store user information. User emails and passwords must not be exposed from the server, moreover, the client doesn't need any information about the users when in offline mode. In online mode, it could easily receive this information from the server. The usernames and emails must be unique.

Coviator_Trip This is the junction table for normalizing the many-to-many relation between Coviator and Trip tables.

Trip The Trip table itself contains only primary ID and trip name, but it has one-to-many relations with both Place and Category tables. The picture, that is in the domain model will be stored beyond database, but its address will be composed from trip ID.

Category Apart from primary ID, name and color, the Category table contains the foreign key to Trip table that defines their relationship.

Place The domain model was defining place coordinates as single parameter, but in practice, it was decided to break it to latitude and longitude for easier storing. The picture will be stored outside of database, but its address will be composed of place ID and trip ID to which this place belongs. The place entity has many-to-one relationship with Trip and Category tables. Another difference from domain model, it was decided to store "created_by" as simple VARCHAR, because this binding is redundant.

Visited_Coviator_Place The junction table to store many-to-many the relationship between coviators and places they have visited.

2.4.2 Client

Almost all information for user interaction the application will receive from local database. The server only provides synchronization and data updates.

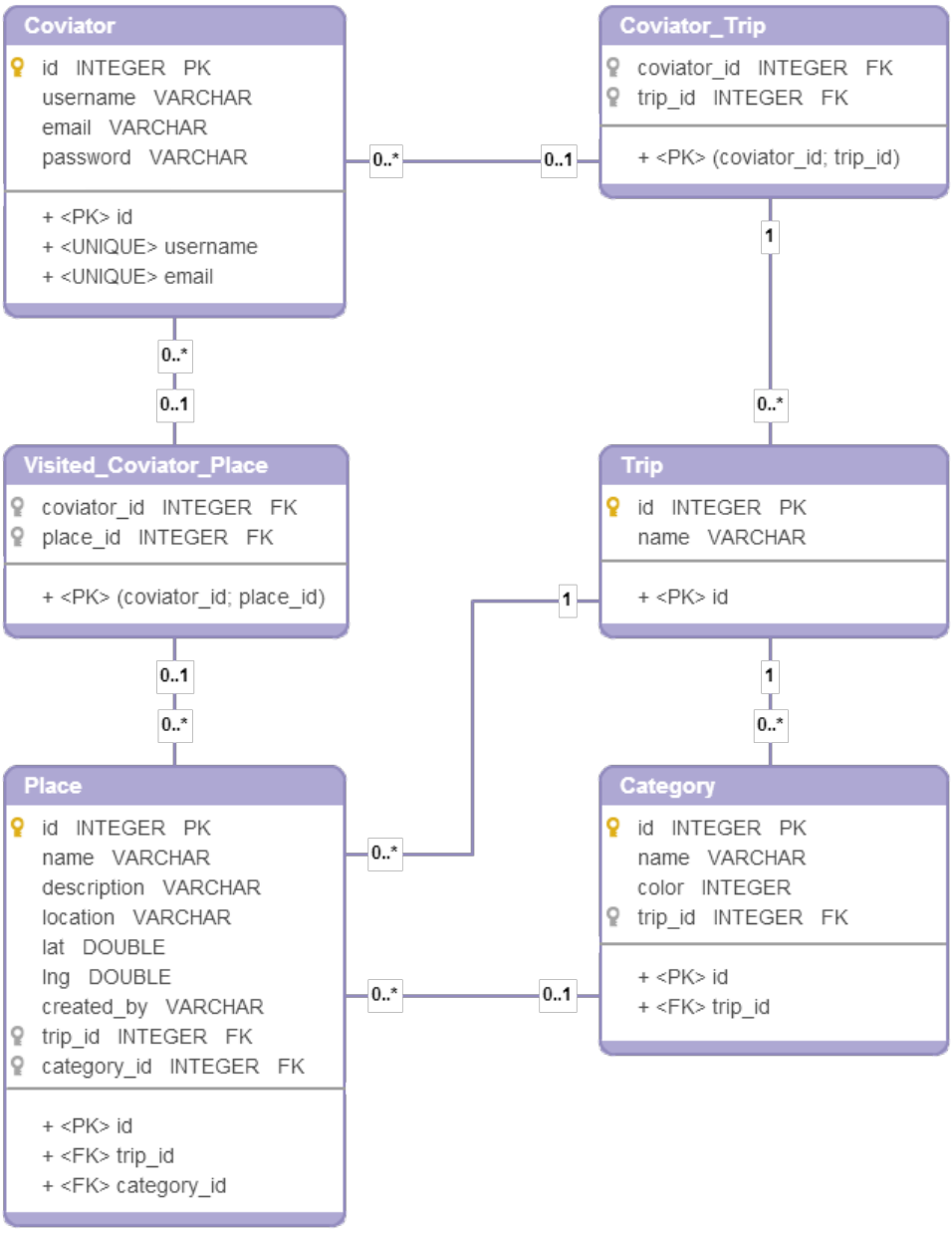


Figure 2.2: Server database model

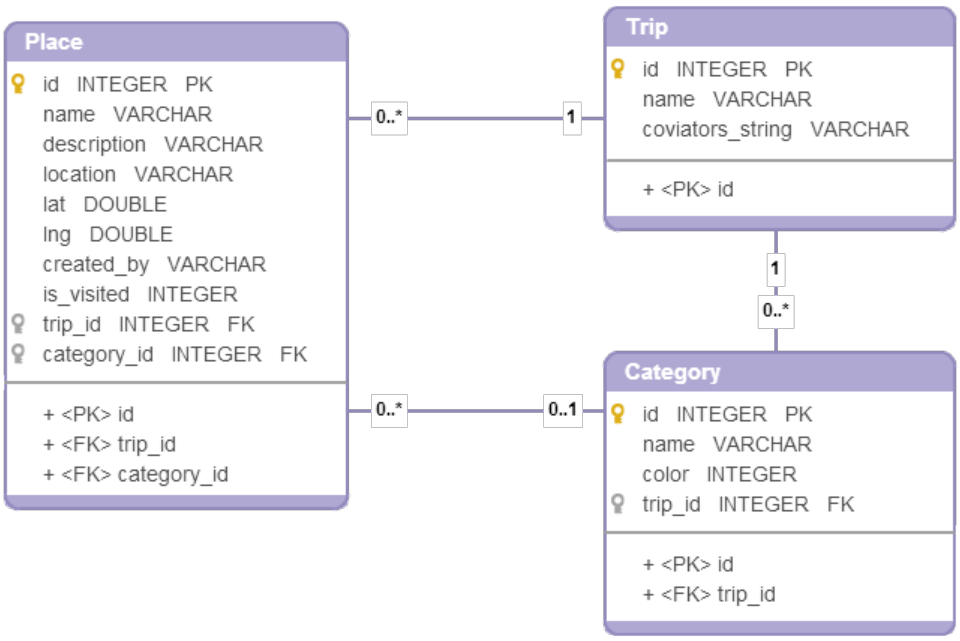


Figure 2.3: Client database model

It was taken into account and client database was attempted to made to answer the presenter layer needs. Detailed client database model is depicted on figure 2.3.

Trip The Trip table on the client side stores only the trips the user belongs to. During synchronization, if some trip was updated, deleted or the user was invited in another trip, te local trip table is updated. Because local database doesn't have users database, the "coviators_string" parameter was added to trip table. It represents the textual form of coviators taking part in particular trip. This parameter is needed because the trip should display at least usernames of coviators who travel with the user.

Category The category table is completely the same as in server database.

Place The place table doesn't differ much from server variant as well. The only difference is that it lacks "visited" relationship with Coviator entity. But it doesn't matter, because client requires "visited" state only for the user itself. That's why boolean parameter "is_visited" is sufficient. SQLite for Android doesn't have native boolean type, that's why it is presented as integer.

2.5 Activity diagrams

In the section 1 of chapter 2 we have described the application behavior from the user point of view. Unfortunately, use cases and even scenarios don't describe the application behavior from the system point of view. All the activities involving client and server sides might be quite complicated since the client works with two data sources: local database and REST API. It is a good idea to show these applications flow with the help of activity diagrams.

In Unified Modeling Language (UML), an activity diagram is a graphical representation of an executed set of system procedures and considered a state chart diagram variation. Activity diagrams describe parallel and conditional activities, use cases and system functions at a detailed level. The purpose of the activity diagram is to model the procedural flow of actions that are part of a larger activity. In projects in which use cases are present, activity diagrams can model a specific use case at a more detailed level. However, activity diagrams can be used independently of use cases for modeling a business-level function, such as buying a concert ticket or registering for a college class [9, 10].

Below are the main activity diagrams of our application. In the figure B.2 the login/register flow is presented. The figure B.3 depicts the actions required to show data. It covers both trip list and trip detail cases because the flow is the same. Two next figures: B.4 and B.5 present the trip and place creation flow respectively. The category creation flow isn't introduced here because it is just a simplified version of the latter. The edition flow will look almost the same. Other activities diagrams are much more plain, so it was decided not to confine ourselves with these four.

2.6 UI/UX design

Up to this point, we were dealing with our application in general. We've came up with structural and behavioral aspects of both the server and the client side of the system and defined the communication rules between them. But the main and the only actor here is the user. Even perfectly designed travel application worth nothing if the user couldn't interact with it. Or if the interface is designed so poorly that complicates user interaction instead of simplify it.

Firstly, it is useful to distinguish between the UX and UI design definitions. UXD (User Experience Design) handles every aspect of the user's interaction with a product, service, or company that make up the user's perceptions of the whole. User experience design as a discipline is concerned with all the elements that together make up that interface, including layout, visual design, text, brand, sound, and interaction. UE works to coordinate these elements to allow for the best possible interaction by users [11]. At the same time, the UID (User Interface Design) seems to be a bit narrower concept. It focuses

on anticipating what users might need to do and ensuring that the interface has elements that are easy to access, understand, and use to facilitate those actions [12].

> *If you imagine a product as the human body, the bones represent the code which give it structure. The organs represent the UX design: measuring and optimizing against input for supporting life functions. And UI design represents the cosmetics of the bodyits presentation, its senses and reactions.* [13]

We'll begin with the task analysis, discuss some details concerning platform-specific development and come up with application wireframes.

2.6.1 Task analysis

Even if use cases and tasks seem to have similar purpose, there is a thin line between them. Use cases describe particular cases of interaction between the user and the system that may be then decomposed to actions and scenarios. But these actions aren't necessarily performed by the user. On the contrary, the tasks are describing the actions, even the minor ones, that user should perform or cause to achieve some goal.

Task analysis analyses what a user is required to do in terms of actions and/or cognitive processes to achieve a task. A detailed task analysis can be conducted to understand the current system and the information flows within it. These information flows are important to the maintenance of the existing system and must be incorporated or substituted in any new system. Task analysis makes it possible to design and allocate tasks appropriately within the new system. The functions to be included within the system and the user interface can then be accurately specified. But generally, task analysis is a fundamental methodology in the assessment and reduction of human error [14, 15].

As we said earlier, task analysis describes the system from user perspective. It takes use cases as input, then derives all the underlying tasks an subtasks, even the minor ones, that user should perform in the scope of particular use case. The first thing is just to write down all these tasks that come to mind. They can have different importance and granularity, but it doesn't matter in this first step.

- Display sign in form
- Sign in
- Display register form
- Register

- Display trips list
- Display create/edit trip form
- Display add coviator dialog
- Display coviator autocomplete suggestions
- Add coviator to trip
- Remove coviator from trip
- Add trip picture
- Delete trip picture
- Submit trip creation/edition
- Display delete trip confirmation dialog
- Confirm trip deletion
- Go to navigation menu
- Display places list
- Display create/edit place form
- Display choose location map screen
- Display location autocomplete suggestions
- Choose location on map
- Confirm location choice
- Choose category
- Submit place creation
- Display places map
- Navigate to place on map
- Go to user location
- Display place detail
- Change place visited state
- Display delete place confirmation dialog
- Confirm place deletion

- Add place picture
- Delete place picture
- Choose filters
- Display categories list
- Display create/edit category form
- Submit category creation
- Display delete category confirmation dialog
- Confirm category deletion
- Enter place search query
- Clear place search query
- Sign out
- Display app info
- Go back
- Show error message

It is good practice sometimes to group tasks by some parameter or to sort them. But in this example, the author found it hard to come out with really useful classification, so it was decided to keep the task list as it is. What the author found really useful is to transfer these tasks onto task graph. Task graph isn't a standardized UML diagram unlike the others in this chapter, but it helps to structure tasks dependencies and can be the basis for wireframes design. The task graph is presented in the figure B.6. Some tasks aren't depicted as graph nodes (go back, show error message), because these actions should be available from multiple parts of the application.

2.6.2 OS Android UI elements

Since its birth, Android application market was a total hell in the sense of average app UI comparing to iOS. The approvement requirements are much weaker and till this moment, design guidelines weren't really good and the majority of developers weren't following them. Moreover, each mobile manufacturer thinks it's good to alter native Android design with their own mods. All this mess doesn't improve user experience.

But, in these latter days, Google makes huge steps to solve this mess and to make Android design unified, clear and user friendly. In author's opinion, they've really made a breakthrough with their last version (5.0 Lollipop) and

introducing of the "Material design" concept. But this isn't just version-specific concept, it is whole design language for different platforms. Material design is a unified system of visual, motion, and interaction design that adapts across different devices. Material design is inspired by tactile materials, such as paper and ink. Material surfaces interact in a shared space. Surfaces can have elevation (z-height) and cast shadows on other surfaces to convey relationships [16].

Let's come up with a little brief on UI elements will be used in "Coviator" application. We'll try to follow Material design guidelines [17] when it is possible.

App bar App bar, or former action bar, isn't a new element. But in Material design it was properly defined. App bar is a special kind of toolbar thats used for branding, navigation, search, and actions. To the left of the app bar there is nav icon that is used either for navigation drawer opening or as a back button. In our example, it will be used for both, depending on the accessibility of navigation drawer. The title in the bar is reflecting the current screen. The icons to the left are optional screen-dependent menu items. We well using them for example for opening filters, create trip form or search. App bar will be present on all screens of our app except login screen (is redundant here) and location choosing map screen, where it will be replaced by Google Maps-styled search toolbar.

Navigation drawer Navigation drawer is a common pattern found in Google apps. It is a sliding panel that displays the apps main navigation options on the left edge of the screen. It is hidden most of the time, but is revealed when the user swipes a finger from the left edge of the screen or, while at the top level of the app, the user touches the app icon in the action bar. It was decided in our app, that navigation drawer will be accessible only from the central nodes of the app: trips list, places list and places map screens. All other screens are leaf nodes, and the back button will be sufficient there. Moreover, the navigation may be overcomplicated otherwise due to possibility of infinite back stack growth. Generally, the navigation drawer contains of the header and the list. Design guidelines recommend not to change drawer content depending on the screen. In our app, we'll leave the list immutable, but the header will change depending on navigation level. If it is trip list, the username will replaced in the header. If it is trip detail, the header will contain trip picture with the "Edit" button. Such solution allows not to change the drawer list and at the same time make trip edition accessible from the detail without encumbering app bar. The options in navigation drawer list allow to access first level of navigation (trip list), display app info and log out. Moreover, in case of extending the

app functionality and navigation hierarchy, it will be easy to implement these new options inside navigation drawer.

Floating action button Floating action buttons, or simply FABs are used for a promoted action. They are distinguished by a circled icon floating above the UI and have motion behaviors that include morphing, launching, and a transferring anchor point. Only one FAB is recommended per screen to increase its prominence. It should represent only the most common action. We are decided to use FAB in several screens. On places map and places list screens, FAB is used to create a new place. It is common action, so its use is justified. On the choosing location map screen, the FAB is appearing after some location is chosen and denotes location confirmation. On places map, we decided to place the second FAB even if it is against guidelines. But such behavior seemed logical to the author, moreover, the official Google Maps app has two FABs too. The second FAB will be "my position" FAB. When the place is chosen on the map, the "add place" FAB will be transformed into "navigate" FAB. These two FABs will perform two familiar functions: navigation to place and moving the map to user location.

Bottom drawer Bottom drawer is a sliding panel, that can be pulled out from the bottom. This panel is used on the map to display place detail information. It has three states: hidden, collapsed and expanded. The panel is hidden (or invisible) when no place is chosen. Collapsed panel appears when user chooses a place. It contains the main information: place title, category, address and distance to the place. Then the user may pull out the panel to see all the information about the chosen place. It isn't the Material standard element. There are bottom sheets in Material design guidelines that are similar visually, but they're intended only to present a simple set of actions. Anyway, we decided to use this pattern, because Google Maps app uses it and it will appear familiar to the users for sure.

Dialogs A dialog is a small window that prompts the user to make a decision or enter additional information. A dialog does not fill the screen and is normally used for modal events that require users to take an action before they can proceed. Material guidelines advise to use dialogs sparingly because they are interruptive in nature. Their sudden appearance forces users to stop their current task and refocus on the dialog content. Not every choice, setting, or detail warrants interruption and prominence. But in most cases, we will use dialogs just for confirmation of sensible actions like deletion or log out. Only one more complicated custom dialog will be used in the app. It is coviator choosing dialog. It was decided that the whole separate screen will be redundant for this task, and the dialog will be more handy. Furthermore, the dialog

doesn't occupy the whole screen and make the user see what coviators he already chosen. The last one dialog we will use is information dialog in offline mode. It will inform user that creation, edition and deletion aren't allowed without Internet connection.

Text fields A text field allows the user to type text into your app. It can be either single line or multi-line. Touching a text field places the cursor and automatically displays the keyboard. The Material text field provides additional features. Floating label shows the hint when the user starts typing. Bottom helper text is used to show the user the errors or the limits on characters. We won't use the character counters in the app, but the errors will be always displayed below relevant text fields. Moreover, text fields can use the autocomplete function, when relevant suggestions are displayed as a list below text field when the user is typing. This feature will be used in coviator and location searches.

Lists and cards Lists are representing a collection of data. Lists are best suited to presenting a homogeneous data type or sets of data types, such as images and text, optimized for reading comprehension with the goal of differentiating between like data types or qualities within a single data type. In our application, we are using lists to represent places, categories and coviators inside trip. But trip items contain more information. The guidelines are stating that if more than three lines of text need to be shown in list tiles, use cards instead. So, it was decided to use cards instead of simple list items to represent trips. Besides, list controls will be used to represent filter items: radio buttons for single choice and checkboxes for multiple choice.

2.6.3 Wireframes

Now, we have all we need to design wireframes, the last step before we could start with prototype implementation. A wireframe is a two-dimensional illustration of a pages interface that specifically focuses on space allocation and prioritization of content, functionalities available, and intended behaviors. For these reasons, wireframes typically do not include any styling, color, or graphics. Simply said, a wireframe is a simple sketch of a user interface [18, 19].

A wireframe can be created digitally, by special software and with details. But the main advantages of wireframing that it's quick and everyone can do it. So, the pen and the paper are the best instruments that help us draw wireframes. Paper wireframes are quick to edit, and allow to reveal drawbacks on earlier stages of design. In the figures B.7 and B.8 the last version of application wireframes is presented. It took the author more than ten attempts to come up with this one, but each new version helped to find some lacks in

the design and improve it at significantly lower costs than it could be on later phases of the project.

Chapter 3

Implementation

3.1 Android client

3.1.1 Tools and libraries

3.1.1.1 Android Studio and Android SDK

The applications for Android operating system are mostly written in Java programming language. But plain Java isn't sufficient for Android development. Android SDK (software development kit) is a mandatory tool for creating actual Android applications. The Android SDK is a set of development tools used to develop applications for Android platform. The Android SDK includes the following [20]:

- Required libraries
- Debugger
- An emulator
- Relevant documentation for the Android application program interfaces (APIs)
- Sample source code
- Tutorials for the Android OS

Every time Google releases a new version of Android, a corresponding SDK is also released. To be able to write programs with the latest features, developers must download and install each versions SDK for the particular phone.

Although the SDK can be used to write Android programs in the command prompt, the most common method is by using an integrated development environment (IDE). During the last year Android Studio has became the

absolute number one IDE for Android development and the official IDE for Android application development. Unlike Eclipse, Android Studio is created by Android developers particularly for Android development and contains tons of useful features without anything unnecessary. Android Studio is based on IntelliJ IDEA software, and on top of its functions, it offers [21]:

- Flexible Gradle-based build system
- Build variants and multiple apk file generation
- Code templates to help you build common app features
- Rich layout editor with support for drag and drop theme editing
- ProGuard and app-signing capabilities

Android Studio has much more different features, but their presentation is out of scope of this project. In short, Android Studio simplifies development process greatly and allows to manage the app easily on different stages.

3.1.1.2 Git

As a revision control system(RCS) it was decided to use Git. Revision control is necessary when working in teams, but even for local use, it is a very useful thing. Revision control systems track and provide control over changes to source code and records changes to a file or set of files over time so that you can recall specific versions later. Git is the most popular modern revision control system. Git is a distributed RCS, working on the peer-to-peer principle, which means that each peer's working copy of the codebase is a complete repository. In our example, when the author is working on the project by himself and has only local repository, it may seem redundant. But the project could always grow and expand, or new developer may be assigned to it, so if the Git repository will be maintained from the start, it can simplify the development process in the future. Moreover, basic Git functions are quite easy to understand and use. Git is often used from the command line, but Android Studio has the support of the main Git functions straight in the IDE.

3.1.1.3 Android support library

The Android Support Library package is a set of code libraries that provide backward-compatible versions of Android framework APIs as well as features that are only available through the library APIs. Our application should support API level 15 and higher, that's why Android support library is used.

3.1.1.4 Dagger

Dagger is a fast dependency injector. It doesn't use reflection, and therefore, works really quickly and doesn't consume additional memory. It provides field injection mechanism which is really useful when working with Android components that doesn't allow to alter the constructors. Dagger suits very well for the implementation of MVP (model-view-presenter) pattern.

3.1.1.5 Otto event bus

Otto is an event bus designed to decouple different parts of the application while still allowing them to communicate efficiently. Otto is working on the post-subscribe principle, when poster and subscriber don't know about each other existence and communicate using special event objects. Otto us useful for handling asynchronous communication between components.

3.1.1.6 Picasso

Picasso library simplifies the downloading of images from different sources straight into views. It handles the callbacks and maintains the cache. It allows as well to apply custom transformations to downloaded images.

3.1.1.7 Retrofit

Retrofit is the library for implementing the REST client on Android device. Retrofit establishes the connection, creates HTTP requests, sends data and handles the callbacks.

3.1.1.8 Butterknife

Butterknife is the view injection library. It allows to avoid tons of boilerplate code while getting the views and assigning listeners to them

3.1.1.9 Other open source libraries

There were used some other smaller libraries, mostly for different Material views.

- **Android Pager Sliding Tab Strip** - Material design tabs panel.
- **Material Drawer** - Material design navigation drawer, sliding from the left.
- **Android Sliding Up Panel** - drawer sliding from the bottom that was used on map screens.

- **Floating Action Button Library for Android** - Material design floating action button.
- **Material Edit Text** - Material design text fields with floating labels and helper texts.

With the release of Android support library version 22.0.0, the design support library was introduced. It contains almost all Material components that the used library have. But at the moment of the app implementation, this version wasn't yet released.

3.1.2 MVP pattern

The big problem of Android applications structure, that it doesn't have clean separation between presentation and logic. Most of the applications use at best the View-Model architecture. As a result, the logic is crammed into Activities and Fragments and programmers have to fight the View complexities instead of solving their business tasks.

The Model-View-Presenter (MVP) pattern or its analogs (MVC, MVVM) is quite popular on some platforms, but in Android development, this is a new phenomenon. Unfortunately, the flaw above is already imposed by Activity/Fragment design. These components, views by nature, contain some presenter functions. Moreover, these classes are quite enclosed, and there weren't much technologies allowing to efficiently implement the MVP pattern. Bu recently, some of them have appeared: dependency injection with Dagger and RxJava are modern tendencies that allow to make MVP not only usable, but very useful as well.

MVP pattern decouples Views from data access mechanisms. As a result, our components are:

- Reusable
- Replaceable
- Easier to test and debug
- Breakable to separate tasks
- Clear to understand

The main drawback of MVP is some boilerplate code. Instead of one Activity or Fragment it takes the programmer to create at least two interfaces and two classes. But let's define the main components of Android MVP first.

View View is a layer that displays data and reacts to user actions. On Android, this could be an Activity, a Fragment, an android.view.View or a Dialog.

Presenter Presenter is a layer that provides View with data from Model. Presenter also handles background tasks and decides what happens when you interact with the view.

Model Model is a data access layer such as database or remote server. The Model is communicating with the other components through Interactors, its interface classes.

When developing for Android, the programmer should always take the components lifecycle into account. In this approach, Presenters are always synchronized with View lifecycle. The View obtains Presenter reference through dependency injection, then, in onResume method, it registers itself to Presenter. The Presenter on the other way, already has the reference to needed Interactors via dependency injection. The Interactors don't care and know nothing about neither Views nor Presenters. They are totally decoupled from Android components lifecycle and communicating with Presenters through posting events on event bus. Presenters are subscribed to appropriate events and can handle them. When Presenter receives events destined to it, it handles it and decides what to show to the user depending on its view state. Such architecture allows to replace or edit Interactors easily without any changes in Views, test each component independently and ensure the lifecycle will be safe even during asynchronous communication.

This subsection was written with the help of [22, 23].

3.1.3 Data access

3.1.3.1 Client database

Android uses SQLite as a database technology. SQLite is an Open Source database. SQLite supports standard relational database features like SQL syntax, transactions and prepared statements. SQLite is embedded into every Android device. Using an SQLite database in Android does not require a setup procedure or administration of the database. You only have to define the SQL statements for creating and updating the database. Afterwards the database is automatically managed by the Android platform. Access to an SQLite database involves accessing the file system. This can be slow. Therefore it is recommended to perform database operations asynchronously [24].

The database can be accessed directly, but it's a good practice to extend ContentProvider class to use as a facade. Content provider is mainly used to expose content to other applications, but it has its advantages even when used inside single application. Content provider is accessible through ContentResolver class from almost everywhere in the app (especially when we can inject Context through dependency injection). Content provider handles closing and locking of database and simplifies the access with URIs. In our

example, DataProvider class extends ContentProvider and provides an API for creating, deleting and updating data in SQLite database.

Content provider here represents the Model layer but its interface isn't convenient for direct communication with Presenter. So, the DBInteractor class serves as an interface between Presenter and Model layers. DBInteractor communicates with DataProvider on worker threads and loads data with custom Loaders. Then it handles callbacks and post appropriate events containing received data on the event bus. All the database interaction is performed asynchronously without interfering with the UI thread.

3.1.3.2 Remote server

As it was said earlier, the application uses Retrofit library to handle REST communication. The CoviatorApi interface maps HTTP requests and API endpoints to corresponding Java methods. Then, the CoviatorApiInteractor is communicating asynchronously with the server through CoviatorApi interface and posts appropriate events to the event bus. If the server is unavailable, the device is disconnected from the Internet, or some other error is occurred during the communication, CoviatorApiInteractor posts the event with appropriate error to the event bus and the Presenter can then show the user what is just happened.

3.1.3.3 Data loading process

The application should work in offline mode, but it should synchronize its data with the server if the device is connected to the Internet. That is why loading requests to the local database and to the server have to be done independently. Even if the application will fail to get the data from the server, the data from the local database will be loaded anyway. But when the data from the server will be received, it will be updated immediately in the local database and shown to the user. This process is depicted in the sequence diagram in the figure 3.1.

3.1.3.4 Data creating/updating/deleting process

As it was stated earlier, the application will be allowed to create, update or delete any data only when the server will be available. In that case, the data changing request will be sent to the server. Then the server will try to preform an appropriate operation and if something will go wrong, the server will send the error to the client. The API Interactor will handle this error and tell the Presenter to show it to the user. Otherwise, if the server will succeed with the operation, the confirmation will be sent to the client. The client database will be changed only during the synchronization (when the data will be requested).

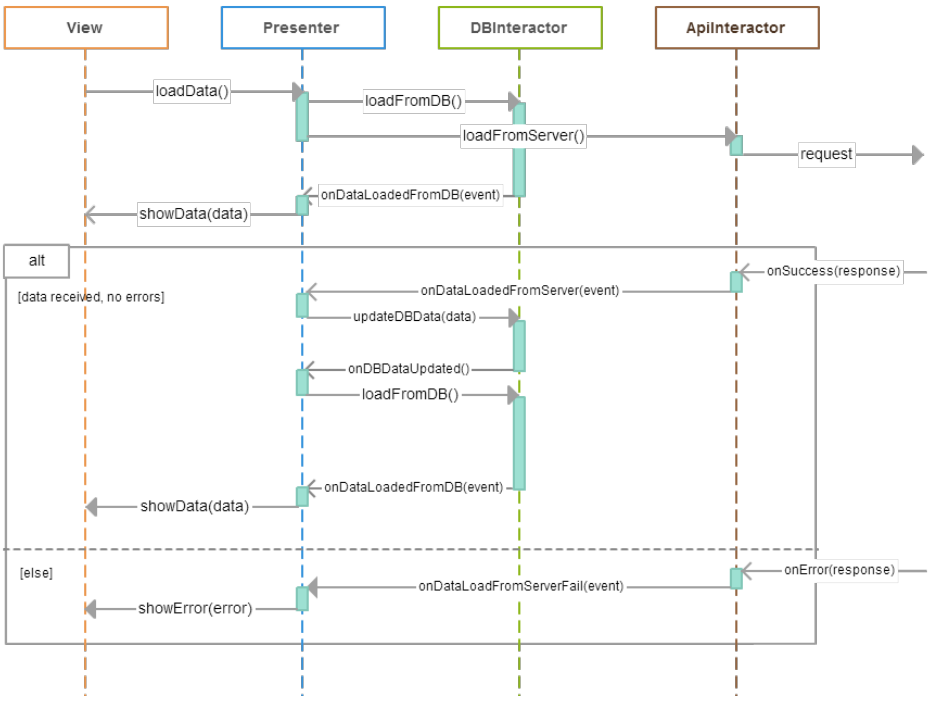


Figure 3.1: Data loading sequence diagram

3.1.4 Packages description

Let's come up with a brief description of packages of the application and classes inside them.

db Contains classes taking care of the database access. DatabaseHelper handles database creation and updates. DataProvider provides the interface to access database operations through URIs.

di Contains dependency injection modules (Dagger library). AppModule and InteractorsModule provide global app dependencies. Each other module provides dependencies for its own scope. For example, LoginModule provide dependencies that are needed during LoginActivity lifecycle only.

domain This package contains Java data objects that are representing the database entities like Trip or Place. The subpackages of "domain" package contain other Model layer objects: events for event bus, custom annotations, filter objects, serializable classes, requests, responses and REST interfaces descriptions.

interactor All the Interactor interfaces and their implementations are located in this package. DBInteractor and CoviatorApiInteractor are already

described in previous subsection. ConnectionInteractor is monitoring Internet connection and notifies other components if the state is changed. GoogleApiInteractor works with Google services like Places API or location services. MediaInteractor is responsible for accessing media resources on the device, storing and retrieving images and getting them from the gallery. SPInteractor is used for storing and retrieving data from SharedPreferences (one of Android persistence technologies).

mvp Contains View layer interfaces that are implemented by different Activities and Fragments. Generally, each view corresponds to some application screen or its element. All the Presenter interfaces and their implementations are located here as well. Each Presenter is bound to corresponding View and serves as a middle man between the View and Model layer (presented by Interactors). All Views and Presenters are extending their base classes that contain common functionality. The Views are attached to Presenters in onResume or similar methods of their Android implementations (Activities and Fragments) and detached in onPause method. It allows to avoid errors related with components lifecycle. For example, if the event from the Interactor is received during screen rotation (when the Activity is destroyed and recreated), the Presenter may refer to the object inside this Activity that doesn't exist. But when the Presenter knows that no View is attached to it, it simply won't do anything or will wait for the View to be attached again.

ui The Android UI components are located there: Activities, Fragments, Adapters and custom Android Views. Most of Activities and Fragments are implementing MVP Views, except the ones that don't have their UI or are just containers. Adapters are considered to be part of View layer and communicating with Presenters of Views they are part of. We won't describe in detail each Fragment or Activity, but generally they represent some application section, screen or part of the screen. Dialogs are also located in this package.

utils Contains different useful static methods and application constants.

3.2 Server

3.2.1 Tools and libraries

3.2.1.1 Play framework

We were already spoken of the Play framework in chapter 2.3. It is a lightweight and fast web framework written in Scala and Java. It allows to map REST endpoints to Java methods in on row only and has "hit refresh workflow". By being RESTful by default, including assets compilers, JSON and

WebSocket support, Play is a perfect fit for modern web and mobile applications. Play follows the modelviewcontroller (MVC) pattern. The Model is the domain-specific representation of the information on which the application operates. The View renders the model into a form suitable for interactions, typically a user interface. The Controller responds to events (typically user actions) and processes them, and may also invoke changes on the model. In a Web application, events are typically HTTP requests: a Controller listens for HTTP requests, extracts relevant data from the event, such as query string parameters, request headers... And applies changes on the underlying model objects [25]. Play also manages database evolutions and automatically applies scripts when database is changed.

3.2.1.2 Ebean ORM

Play framework uses Ebean object-relational mapping for automatically convers Java objects into relational database entities and vice versa.

3.2.1.3 Jackson

Jackson is a multifunctional JSON processor that allows quickly convert JSON strings into Java objects and vice versa.

3.2.2 Packages description

The play framework structure is reflecting MVC pattern and split code into three packages. Our application doesn't have its own View layer because it serves only as mobile client backend.

controllers The Application class is the only class inside this package. All the methods mapped to REST endpoints are located inside it. Each of them accepts the request with optional body and query parameters, performs some operation with database and sends the success or error response depending on the result.

models Contains the main POJOs corresponding to database entities: Coviator, Trip, Place and Category. The junction tables are generated automatically by Ebean ORM. Also, responses and requests Java objects are located here, as well as data validators.

Chapter 4

Testing

4.1 Testing by developer

This kind of testing was performed during all stages of implementation after each new chunk of functionality has been added to the application. It includes UI testing, Android components lifecycle testing, database integrity testing, client-server communication testing and synchronization testing. During development and testing the server was running locally and three devices (LG Nexus 4, LG Nexus 5 and Sony Xperia Z) were available to the author.

A big amount of bugs and errors was revealed during these tests. For example, the crashes of the app while accessing objects inside Android component with finished lifecycle, improper synchronization between local and remote data or bad HTTP requests. Also, there was a problem with adding pictures to trips or places. Big images caused lags in UI interaction. This problem was solved by resizing pictures while loading them to the view. The map was a problem thing as well. Because of complicated asynchronous data obtaining mechanisms, the map didn't show the correct data or was simply crashing. Thanks to the MVP pattern, these sequences were clear easy debuggable. Of course there were much minor bugs, crashes and inconsistencies, most of which were fixed after these tests.

4.2 Usability testing

After the prototype was ready and the server was functional, it was decided to run tests on real users.

> *The most effective way of understanding what works and what doesnt in an interface is to watch people use it. This is the essence of usability testing. When the right participants attempt realistic activities, you gain qualitative insights into what is causing users to*

have trouble. This helps you determine how to improve the design. [13]

Usability testing is the technique that engage users to participate in realistic scenarios. While the user is performing given tasks, the tester is observing and making notes. The main purpose of this kind of testing is to reveal the real obstacles the user could encounter, thus the real problems in UI design. That's why there are some good practices the tester should follow:

- **Make the Task Realistic** Asking a participant to do something that he wouldnt normally do will make him try to complete the task without really engaging with the interface. Poorly written tasks make it harder for participants to suspend disbelief about actually owning the task.

- **Make the Task Actionable** Its best to ask the user to do the action, rather than asking them how he would do it. Otherwise, the participant is likely to answer in words, not actions.

- **Avoid Clues and Describing the Steps** Step descriptions often contain hidden clues as to how to use the interface. It may skew the results and hide the problems which the user may have encountered without your hint. [13]

4.2.1 Usability testing process

The situation gets a little complicated since it's quite hard to simulate the real use scenarios of our application, because shared access functionality requires several users at the time to test it. Then, the scenario may be really time-stretched, because the trip with its planning phase may last for several days or even months. Moreover, it's hard to make the test look realistic when one of the main application functions is orientation on map, and the user is just sitting in the front of your desk. Despite all that, the list of user scenarios was composed with the best attempts to make it realistic and actionable.

- **Task 1** Login to the application. Register if you dont have an account.

- **Task 2** You are planning to visit Prague again, youve already been here, but you didnt have Coviator app with you. Create a trip and invite your friend Marty McFly to it. (*Comment: Marty McFly and several other "coviators" are already registered in the system*)

- **Task 3** You forgot to invite your other friend Doc Brown to your trip to Prague. Do it now! Moreover, its a good idea to add some Prague picture to the trip.

- **Task 4** Add some places to the trip. Add a couple of your favourite restaurants, the place where you stay, your university and some place you love, but remember only its location. Its good to classify some of these places to some categories. But you have to create them first.
- **Task 5** You are in Prague and you are hungry. But you are well prepared. Navigate to the closest restaurant.
- **Task 6** Unfortunately, the quality of this place has worsen and the dessert was awful. You should add a photo of it to your place detail, add some notes and mark this place as visited.
- **Task 7** Now you want to go home, but you dont remember the way. Find this place on the map and try to get there.
- **Task 8** You were almost home when Marty McFly texted you that he found some awesome bar near you and you should immediately be there And that he already added it to your trip. Ok, lets drink one or two with Marty. Find this place on the map. (*Comment: while the user is reading this task, the tester is adding a new place to the trip from the Marty McFly account*)
- **Task 9** Your trip is over. Lets see now what places you haven't visited. But you dont have Internet connection here (simulate data off).

Each user from our testing group was prompted to respond to two little surveys: before the test and after the test. The first survey contains general questions about the user and his experience with Android. The second one is taking feedback on the user experience with our application.

Before-test survey:

1. What is your age? (Under 20, 21-30, 31-40, 41-50, over 50)
2. Do you own an Android device? (Yes / No)
3. If yes, what? (Device model)
4. Are you interested in this application functionality? (Yes, No)

After-test survey: (contains open questions only)

1. Was the navigation in the application clear? What did confuse you?
2. Was it hard to find some particular place or place group?
3. Was the map clear and easy to use? What was confusing?
4. Did something confuse you in the application generally?

5. Maybe some functions were missing to you in the app that you would like to add?
6. Did the app behave as you expected from an Android app? What wasn't familiar?
7. Would've you use this application for your trips? Why?

Seven users did participate in this survey. One of them answered "Under 20" to the age question, one of them has "30-40", and all the last are in the "20-30" category. Six users have their own android device, only one is Windows Phone owner. The devices the respondents use are: Nexus 5, Sony Xperia Z, LG G2, Vodafone Smart Prime 6, HTC Desire HD and another Nexus 5. All the users mentioned that they are quite interested in the application functionality. The tests were performed in a small room without external interference. The users were given their tasks one by one without limitation in time. The tester wasn't allowed to give any clues about UI, but had to answer any other questions. While the users were performing their tasks, the tester was taking notes. Then the users were thanked and asked to answer the after-test questionnaire.

4.2.2 Usability testing results

Now, let's take a look on the results of the testing step by step. Below there are our tasks, their estimate solutions and tester observations.

1. **Task 1** Login to the application. Register if you dont have an account.

   Estimated solution: User enters his username and password and clicks "login". If he doesn't have an account, he clicks "register", fills and sends the form and if everything is correct, the app logs him in automatically.

   Users solution: This functionality is common for almost all Android apps, so everybody passed the step 1 as expected. Only one user mentioned that the app isn't hiding the keyboard after login if it was shown.

   Observer notes and users comments: Nothing special.

2. **Task 2** You are planning to visit Prague again, youve already been here, but you didnt have Coviator app with you. Create a trip and invite your friend Marty McFly to it. (*Comment: Marty McFly and several other "coviators" are already registered in the system*)

   Estimated solution: User clicks "add" button, fills the name of the trip, adds photo if he wants, then finds Marty McFly by name and adds him to the trip. Finally, he presses the "submit" button.

Users solution: There weren't problems with this task as well. Users found the corresponding buttons quickly and proceeded as expected.

Observer notes and users comments: Users were quite familiar with the autocomplete feature. Most of them were waiting for the information to appear after typing one or two letters.

3. **Task 3** You forgot to invite your other friend Doc Brown to your trip to Prague. Do it now! Moreover, its a good idea to add some Prague picture to the trip.

   Estimated solution: User opens the newly created trip detail, opens the drawer menu. Then he clicks on the "edit" button and proceeds to the trip edition screen. If the user has not added the picture to the trip in the previous step, he adds it now. He invites new coviator the same way as before and saves changes with the "submit" button.

   Users solution: Here the first problems begin. More than a half of respondents spent much time trying to find the edit interface. They were confused with the "edit" button location in the drawer menu. The other part was completed without problems.

   Observer notes and users comments: Some users mentioned in the survey that they're expecting the "edit" button anywhere but not in the drawer menu.

4. **Task 4** Add some places to the trip. Add a couple of your favourite restaurants, the place where you stay, your university and some place you love, but remember only its location. Its good to classify some of these places to some categories. But you have to create them first.

   Estimated solution: Click on the "add" floating button. Enter the name and optional description. Choose category from the list or create it with "+" button and choosing the color and the name. Choose location by long click on the map or by choosing the address with the help of autocompletion. Confirm location by clicking the "done" floating button. Add optional picture and save the place with "done" button. Repeat for other places.

   Users solution: All of the respondents have no problem with finding the create interface, filling the form and choose the category. Half of the users have chosen the location by typing the place name, but others were trying to find the place by zooming and panning the map. Some of them weren't then able to figure out how to mark the place. They were trying to click once or two times but most of them eventually understood the long click pattern.

Observer notes and users comments: It turned out that the users aren't completely familiar with the long-click-to-mark pattern despite its wide use in most of the map applications including Google ones. One of the users made an observation that there is no way to clear the place category, only to choose another one or delete the category itself.

5. **Task 5** You are in Prague and you are hungry. But you are well prepared. Navigate to the closest restaurant.

   Estimated solution: Open filters screen from the trip detail by click "filters" button in the app bar. Filter restaurants only by checking the corresponding checkbox. Apply filters and open map by click "map" button in the app bar. Choose the closest marker to user location. Click "navigate".

   Users solution: Only two respondents have used the filters, others either found their restaurant in the list and clicked on it, or clicked the "map" button and found the closest one.

   Observer notes and users comments: Each solution is correct because it achieves the expected result. But this task wasn't representative because of the small number of places in the trip. It is plausible that the more places with different categories will be in the trip, the more users will use the filters to find their place.

6. **Task 6** Unfortunately, the quality of this place has worsen and the dessert was awful. You should add a photo of it to your place detail, add some notes and mark this place as visited.

   Estimated solution: Click on place marker on the map. Swipe up the bottom panel with place detail. Click "edit" button. Add the photo and description as in task 4 and save changes. Click the "visited" checkbox in the place detail.

   Users solution: Only two users have found the place detail immediately. Others have spent some time to find out how to edit the place details. They've tried to find it in the places list, in the filters, tried to use search. But eventually all of them found it.

   Observer notes and users comments: Again the users weren't expected to see the place detail in the sliding bottom panel even if it is the common pattern for almost all map applications including Google. Maybe it is confusing because the main screen of the trip detail is the list screen and the users aren't thinking that the map should contain the information. Or the users don't use the map applications to often to get used to this behavior.

7. **Task 7** Now you want to go home, but you dont remember the way. Find this place on the map and try to get there.

   Estimated solution: Again as in the task 5, find the "home" marker by filters, search or color on the map or in the list and navigate to it.

   Users solution: Most of the users just found the "home" place in the list and clicked it to find on the map.

   Observer notes and users comments: Basically it's the same task as the task 5 except the place is uniquely defined. The users haven't encountered any problems and they seem to act more confidently than before while completing this task.

8. **Task 8** You were almost home when Marty McFly texted you that he found some awesome bar near you and you should immediately be there And that he already added it to your trip. Ok, lets drink one or two with Marty. Find this place on the map. (*Comment: while the user is reading this task, the tester is adding a new place to the trip from the Marty McFly account*)

   Estimated solution: Return to trip detail screen and find the new place either on map or in the list.

   Users solution: Some of the users weren't sure what they should do (they didn't see how the interviewer added this new place to their common trip from another device), but all of them returned to the trip detail screen and either filtered places by the new category "bar" or found the bar straight in the list.

   Observer notes and users comments: The users were positively surprised as the new place quickly appeared in their list.

9. **Task 9** Your trip is over. Lets see now what places you haven't visited. But you dont have Internet connection here (simulate data off).

   Estimated solution: Go to filters screen, filter by "not visited". Apply changes.

   Users solution: Some of the users were wondering how the missing connection may affect this task, but completed it as expected.

   Observer notes and users comments: The lack of Internet connection here didn't make any difference. It was to confuse user if he's still able to use the application.

Lets' summarize our testing results together with the results of the after-test survey. Some of our assumptions about the functionality and the implementation of particular features were confirmed. The users accepted positively

the shared access feature and some of them mentioned the usefulness of this function in the survey. The Android users said that the application has the "Android feel" and they don't need much time to get used to it. The offline mode wasn't tested properly, but a couple of users said that they're expecting this feature in their trip app. Almost all users said that the navigation was quite clear, simple and intuitive except some things that were confusing. The first thing is the edit button for the trip. The navigation drawer isn't the place where the users are expecting to find edit interface. The second thing is the place detail. Not all users are familiar with the Google maps pattern and it took time for them to figure out where it was. The next thing is the long click pattern to mark a place on the map. Not all the users realized it immediately. Then one of the users mentioned that there isn't an "empty" category choice and if some category was chosen for the place, it can't be reset, only by deleting the category. Moreover, there were some minor but useful suggestions from the users that may improve the app too. For example, it is good to hide the keyboard after login if it was visible. Or if there isn't any place on the map, the app should zoom initially to the user current location. Or a pull-to-refresh pattern for the places/trips list. Half of the users said that they're considering to use this app if it will be finished because it is easy to manage the common trip with friends and it works offline. The other half aren't interested in it because they don't travel or the applications they use have more functions.

4.2.3 Applying testing results for further application improvement

Usability testing really helped a lot to reveal the flaws of the current application and find out what features and improvements will make the users experience better. Now, with the help of our usability testing results, let's try to define the course of the future development of Coviator app.

First of all, the errors and the bad behavior should be fixed. It includes some crashes that were revealed by the programmer and the minor problems mentioned in subsection 4.2.2 like the keyboard hiding after finish typing or the option to clear a category for the place. Then, the main issues from subsection 4.2.2 should be considered.

The location of "edit trip" button should be changed definitely. The best of the options is to place it in the app bar like some users mentioned. The problem is that there are already three buttons there and the app bar will be overweight. On the other hand, switching between the list and the map can be extracted to the tabs and viewpager. This will take additional place on the screen, but make the navigation clearer.

It turned out that the place detail is tricky to find. But since the bottom sliding panel is the common place to find it in the most popular map applications, we're decided to leave it there. We just need to help somehow

the users to find it. There are several options. The first one is to make the place detail expanded by default when going to the map by clicking on some particular place in the list. But the user may go to the map without choosing some particular place, so the problem persists. Moreover by the time it may annoy the user that the place detail will be always expanded even if he just wanted to locate it on the map. The better practice is to show the user some hint when he opens the map for the first time. It will make clear for the user what he should do without annoying him for the second time. Moreover, the secondary action is considered for each item in the places list that will allow the user to go to the edit interface straight from the list.

The third testing problem was the long click pattern. But again, this is the pattern supported by the Google itself, so we won't change it. Instead, it is good to show some hint the first time the user chooses the location for a place.

Conclusion

The result of this thesis is an Android mobile application and a server that communicate with each other via REST service. The application allows its owner to create and edit the list of user-defined trips, add places to them and assign to different categories. The application allows user to see these places in the form of the list or in the form of markers on the map. Multiple users of this application may share the same trip and see each other changes real-time. All application functionality except editing and creating new places is available without Internet connection. The application allow user to display places on the map and navigate to them using the current location. The user may filter trip places by user-defined categories and "visited" status. Moreover, multiple other features are implemented in this product: adding photos, description and ratings, marking places as visited and others. Some screenshots are available in appendix B.

For now, the application isn't published on the Google Play store and the server is running only locally, but the publishing is planned in the future after the changes will be implemented.

During this work, all the requirements described in the second chapter were met. The application went through all software design and development steps and the prototype was created. The server and the REST API allowed to synchronize data and provide shared access real-time. When the application became functional, the usability tests were performed. The usability testing confirmed some of the design hypotheses, but revealed several problems and misconceptions that were analyzed and the solutions were discussed.

But more important, this work allowed its author, me, to gain a lot of valuable experience. I was able to go through complete software development lifecycle and understand why all these steps are so important. For the first time I was able to implement both parts of the project, Android client and backend, and design the communication between them. I've learned lots of new things specific for Android: UI, work with database and communication via HTTP. And again for the first time, I was able to conduct a user testing

and draw conclusions from its analysis. I was able to improve my general software development process understanding and specific technical skills.

□

Bibliography

[1] Chekuri, R. Why Windows Phone not Catching up with Rivals. October 2014. Available from: http://techiemadness.com/why-windows-phone-not-catching-up-with-rivals/

[2] Hughes, N. While 91Android hold 10Available from: http://appleinsider.com/articles/14/08/22/while-91-of-apple-users-run-ios-7-five-different-versions-of-android-hold-10-share

[3] Jacobson, I.; Christerson, M.; Jonsson, P.; et al. *Use-Case 2.0, The Guide to Succeeding with Use Cases*. Addison-Wesley Professional, 1992, ISBN 978-0201544350.

[4] Use Cases. Available from: http://www.usability.gov/how-to-and-tools/methods/use-cases.html

[5] Larson, R. Scenarios and Use Cases Useful Techniques. July 2010. Available from: http://www.watermarklearning.com/blog/scenarios-and-use-cases/

[6] Leffingwell, D. Domain modeling. February 2014. Available from: http://www.scaledagileframework.com/domain-modeling/

[7] Hunter, T. Principles of good RESTful API Design. December 2013. Available from: http://codeplanet.io/principles-good-restful-api-design/

[8] Stormpath. Secure Your REST API... The Right Way. April 2013. Available from: https://stormpath.com/blog/secure-your-rest-api-right-way/

[9] Janssen, C. Activity Diagram. Available from: http://www.techopedia.com/definition/27489/activity-diagram

[10] Bell, D. UML basics. Part II: The activity diagram. 2003. Available from: https://www.ibm.com/developerworks/rational/library/content/RationalEdge/sep03/f_umlbasics_db.pdf

[11] Usability Body of Knowledge, Glossary. Available from: http://www.usabilitybok.org/glossary

[12] User Interface Design Basics. Available from: http://www.usability.gov/what-and-why/user-interface-design.html

[13] Lamprecht, E. The Difference Between UX and UI Design- A Laymans Guide. June 2015. Available from: http://blog.careerfoundry.com/the-difference-between-ux-and-ui-design-a-laymans-guide/

[14] Task Analysis. Available from: http://www.usabilitynet.org/tools/taskanalysis.htm

[15] Embrey, D. Task Analysis Techniques. 2000. Available from: http://www.humanreliability.com/articles/Task%20Analysis%20Techniques.pdf

[16] Material design with Polymer, guide. 2015. Available from: https://www.polymer-project.org/0.5/docs/elements/material.html

[17] Google. Material design guide. 2015. Available from: http://www.google.com/design/spec/material-design/

[18] What is a Wireframe? February 2014. Available from: http://experience.sap.com/basics/post-143/

[19] Wireframing. Available from: http://www.usability.gov/how-to-and-tools/methods/wireframing.html

[20] Janssen, C. Android SDK. Available from: http://www.techopedia.com/definition/4220/android-sdk

[21] Android Studio Overview. Available from: http://developer.android.com/tools/studio/index.html

[22] Leiva, A. MVP for Android: how to organize the presentation layer. April 2014. Available from: http://antonioleiva.com/mvp-android/

[23] Mikheev, K. Introduction to Model-View-Presenter on Android. March 2015. Available from: http://konmik.github.io/introduction-to-model-view-presenter-on-android.html

[24] Vogel, L. Android SQLite database and content provider - Tutorial. August 2014. Available from: http://www.vogella.com/tutorials/AndroidSQLite/article.html#overview_sqlite

[25] Play framework: The MVC application model. Available from: https://www.playframework.com/documentation/1.0/main

[26] Group, N. N. Turn User Goals into Task Scenarios for Usability Testing. January 2014. Available from: http://www.nngroup.com/articles/task-scenarios-usability-testing/

Appendix A

Acronyms

API Application programming interface

SDK Software development kit

HTTP Hypertext transfer protocol

JSON JavaScript object notation

REST Representational state transfer

UML Unified modeling language

MVC Model-view-controller

MVP Model-view-presenter

ORM Object-relational mapping

JPA Java persistence API

SSL Secure sockets layer

UI User interface

UX User experience

IDE Integrated development environment

RCS Revision control system

Appendix B

Figures

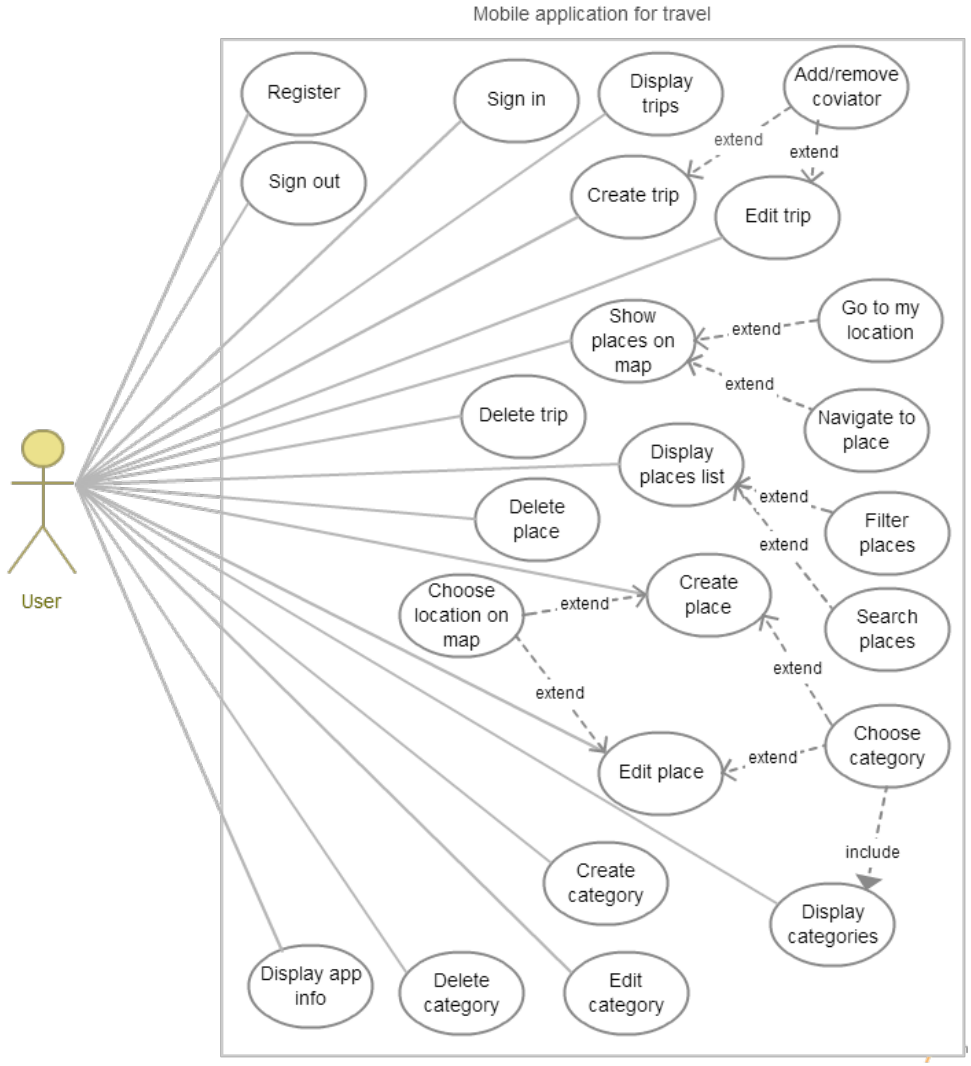


Figure B.1: Use case diagram

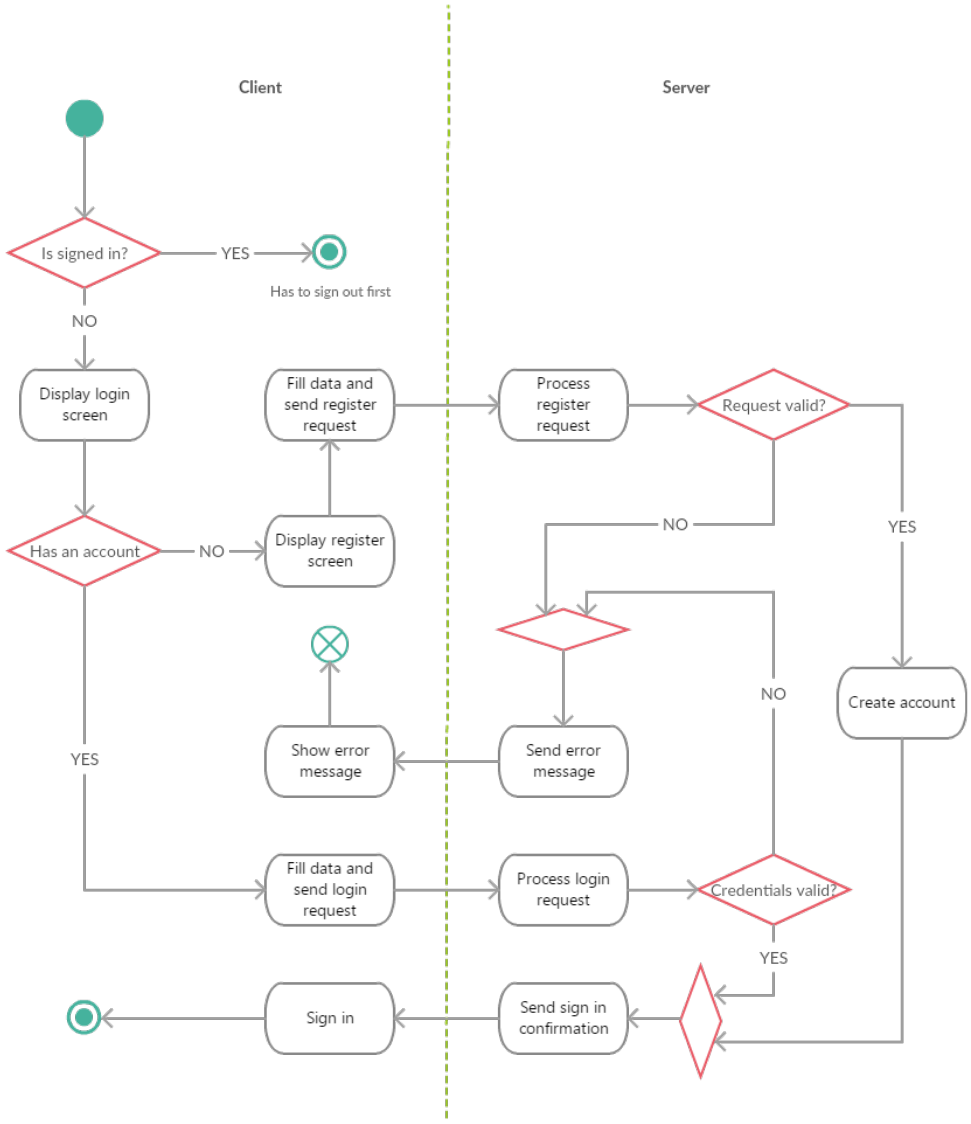


Figure B.2: Sign in/register activity diagram

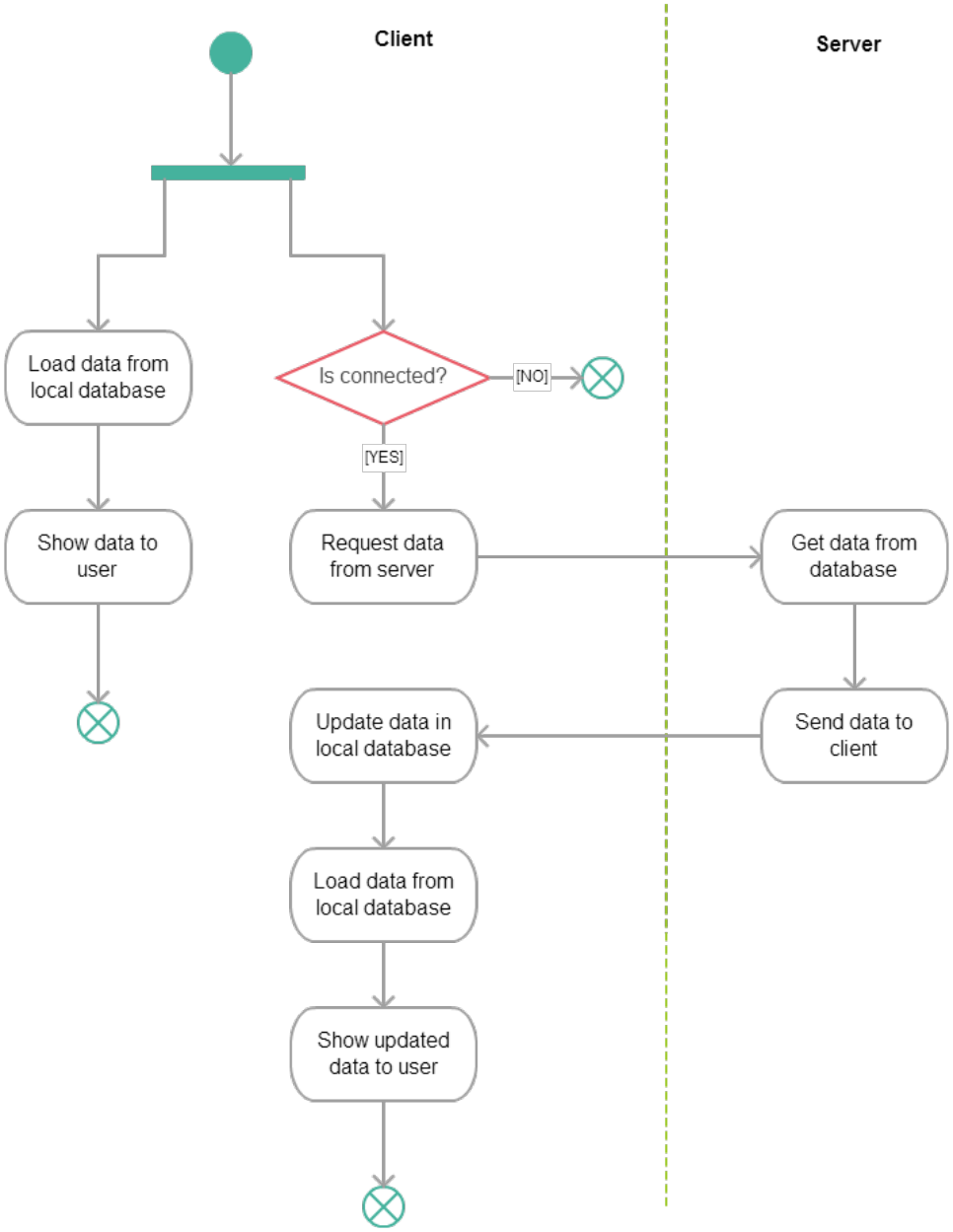


Figure B.3: Display data activity diagram

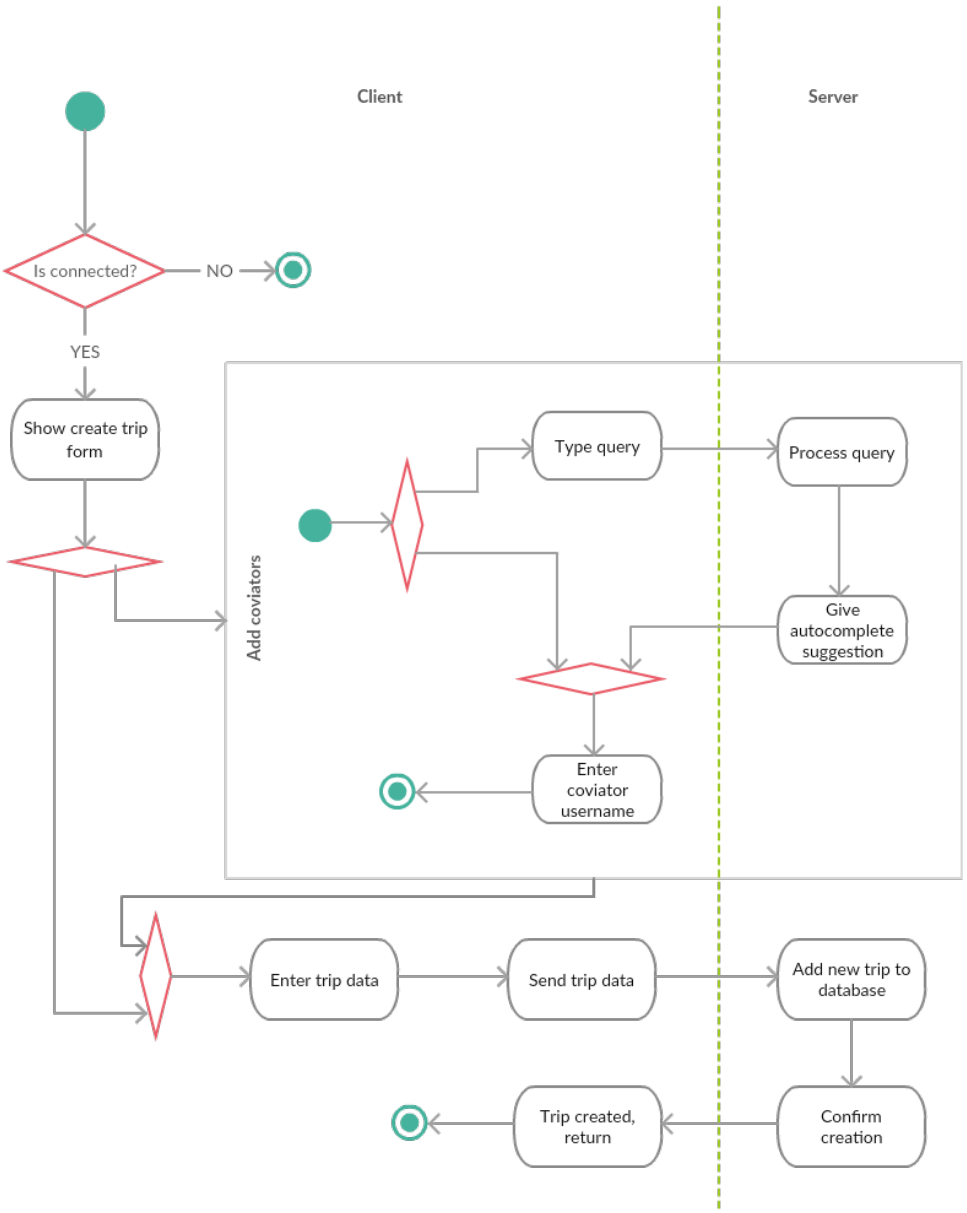


Figure B.4: Create trip activity diagram

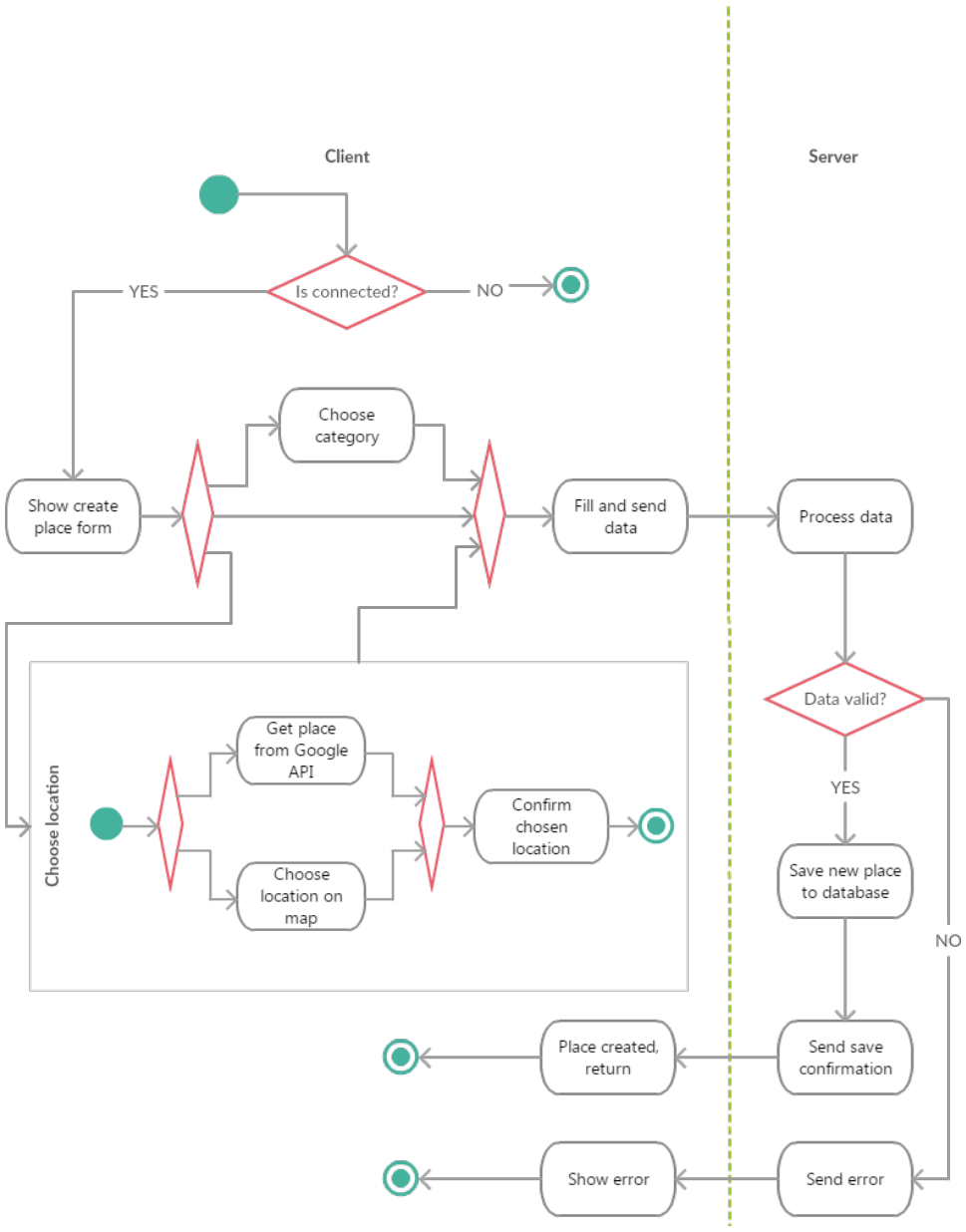


Figure B.5: Create place activity diagram

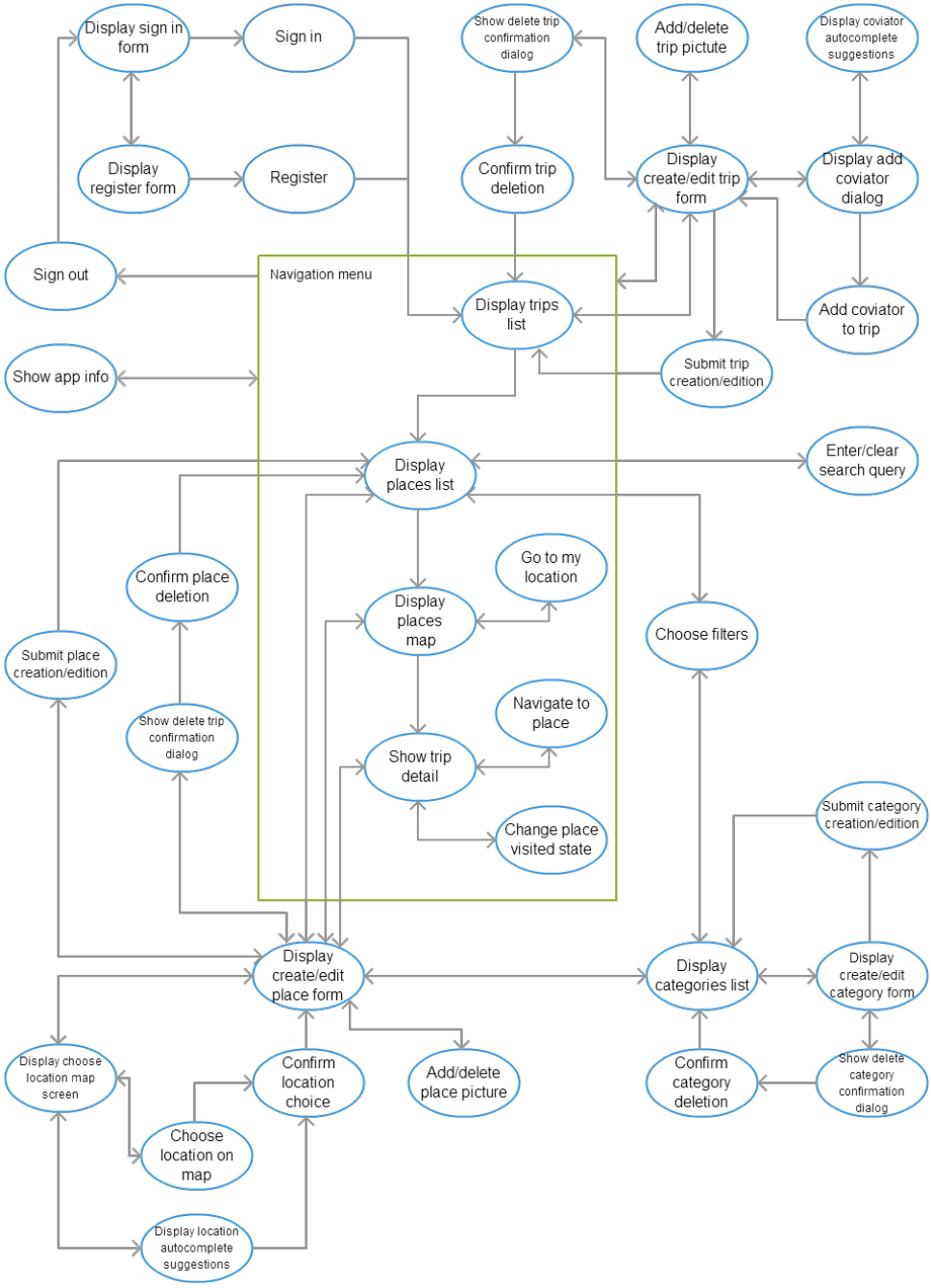


Figure B.6: "Coviator" task graph

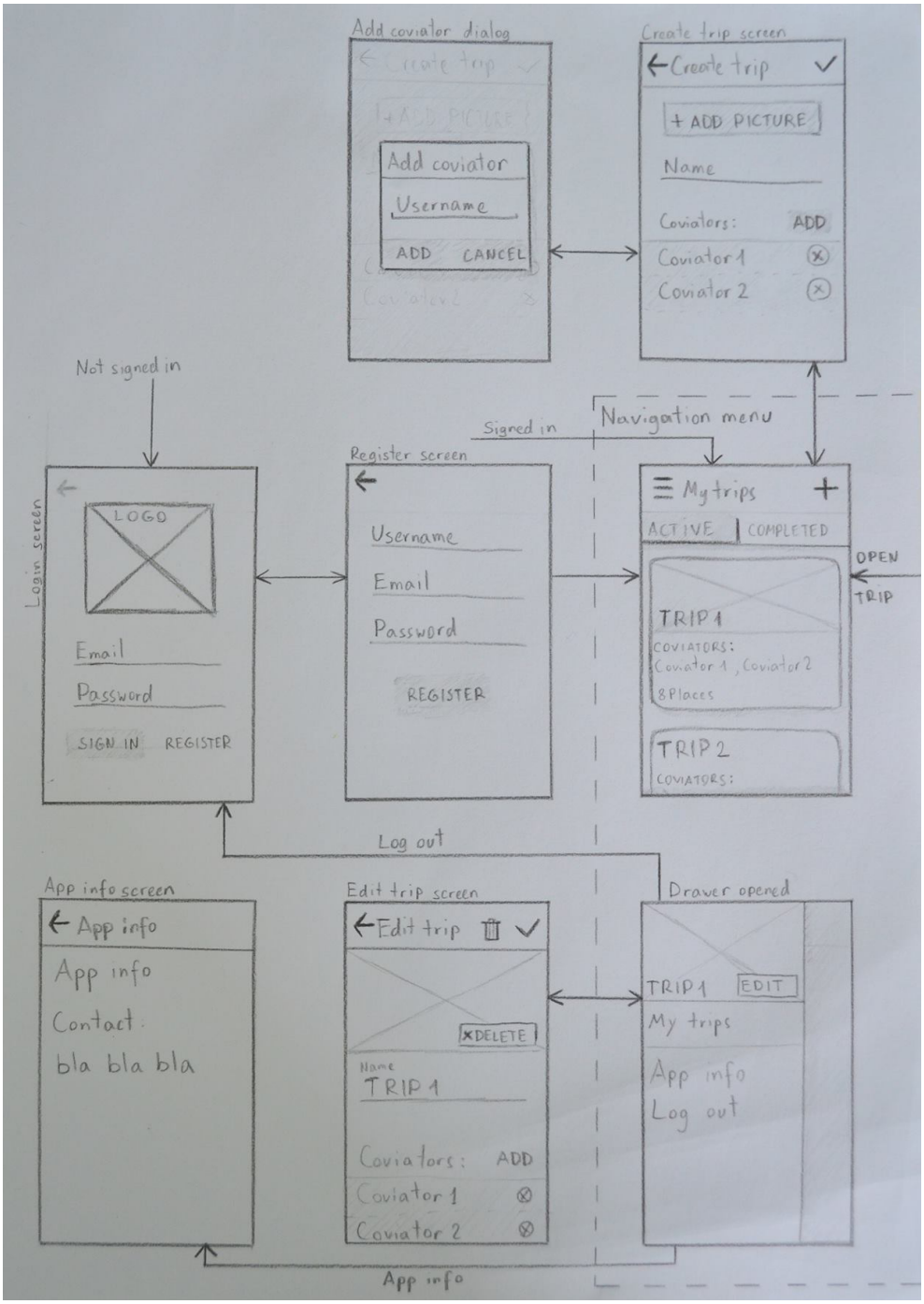


Figure B.7: Wireframes, part 1

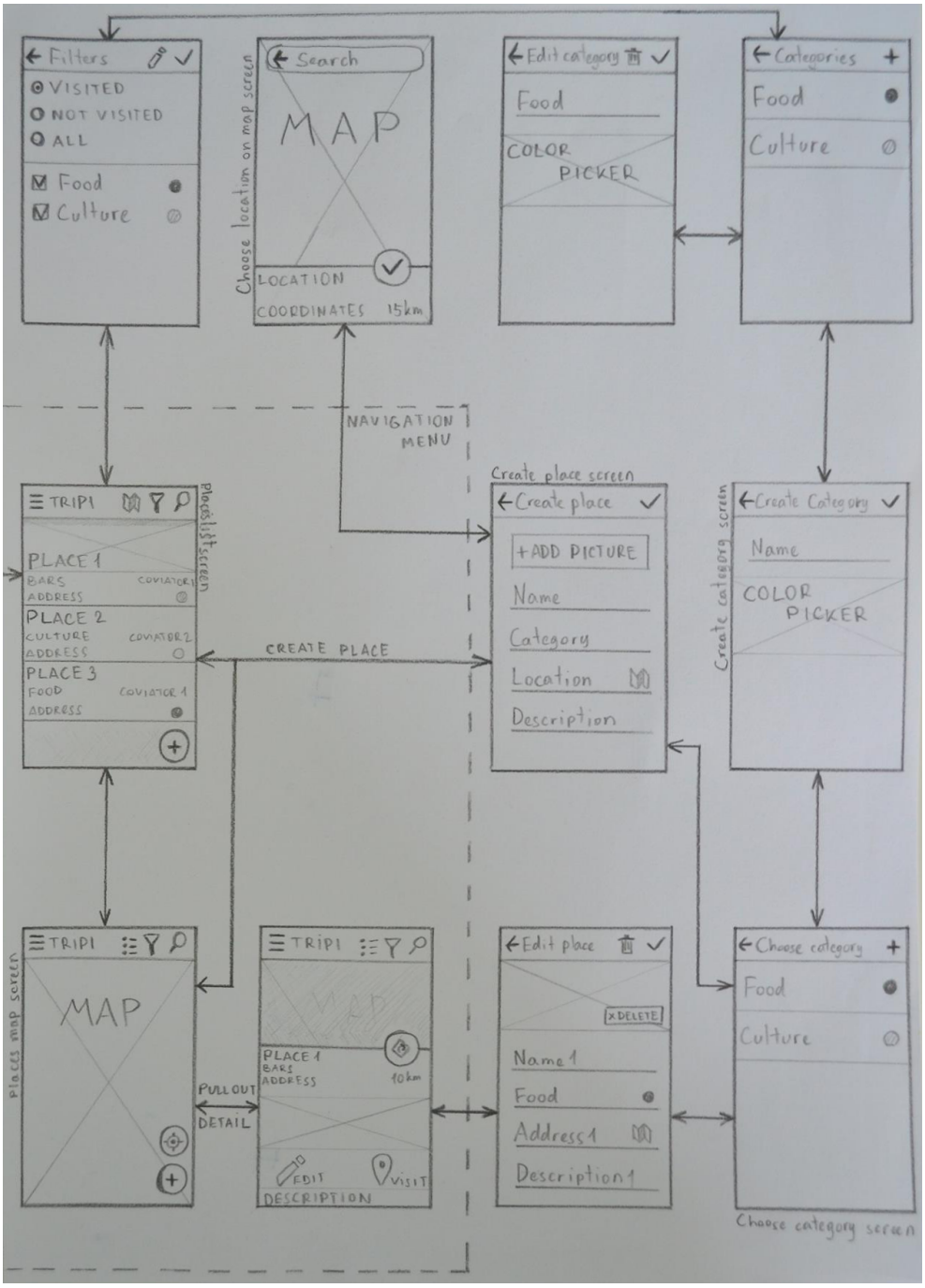
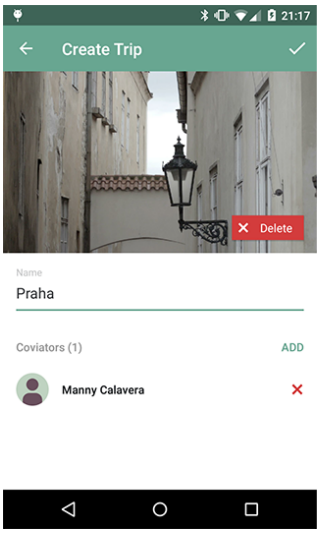
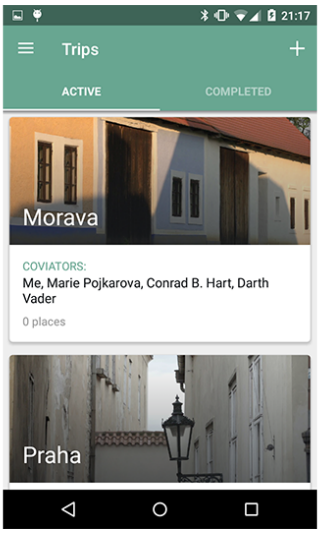


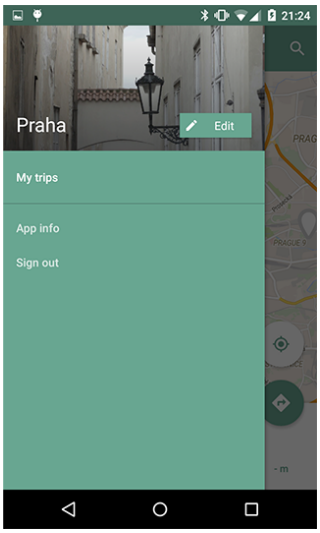
Figure B.8: Wireframes, part 2

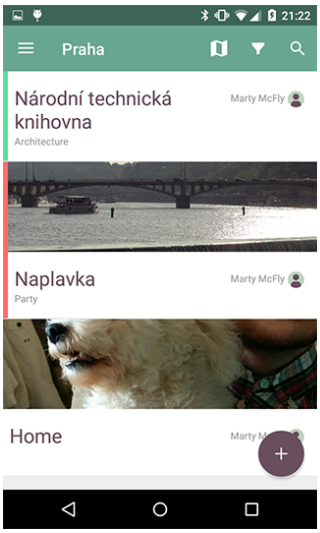
Appendix C

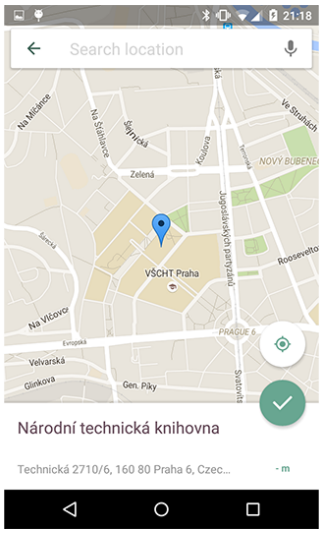
Application screenshots

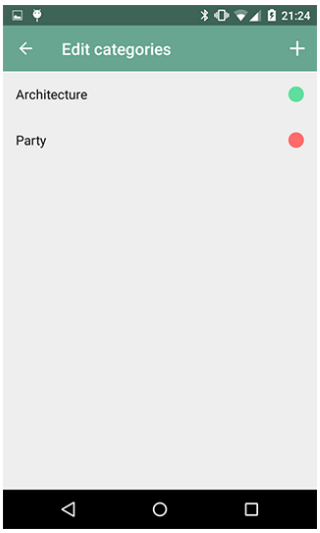


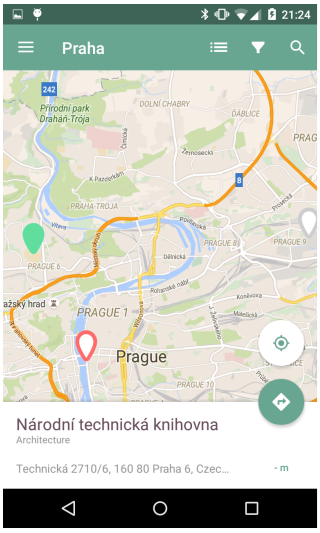


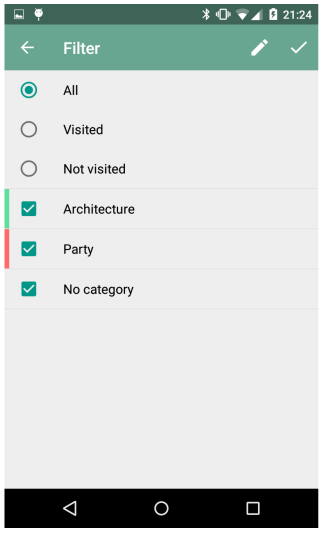


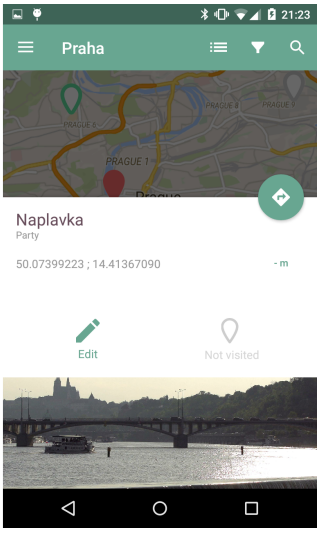

21:22
Praha
Národní technická knihovna
Architecture
Marty McFly
Naplavka
Party
Marty McFly
Home


21:18
Search location
VŠCHT Praha
Národní technická knihovna
Technická 2710/6, 160 80 Praha 6, Czec...


21:24
Edit categories
Architecture
Party


21:24
Praha
Prague
Národní technická knihovna
Architecture
Technická 2710/6, 160 80 Praha 6, Czec...


21:24
Filter
All
Visited
Not visited
Architecture
Party
No category


21:23
Praha
Naplavka
Party
50.07399223 ; 14.41367090
Edit
Not visited

Appendix D

Contents of enclosed CD

- `readme.txt` ..... the file with CD contents description
- `src` ..... the directory of source codes
  - `impl` ..... implementation sources
    - `android` ..... Android client source code
    - `backend` ..... server source code
  - `thesis` ..... the directory of LaTeX source codes of the thesis
- `text` ..... the thesis text directory
  - `thesis.pdf` ..... the thesis text in PDF format