FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Název:	Cestovní seznam pro Android
Student:	Marek Alexa
Vedoucí:	Ing. Vratislav Zima
Studijní program:	Informatika
Studijní obor:	Softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	Do konce letního semestru 2017/18

Pokyny pro vypracování

Analyzujte potřeby cestovatelů při balení zavazadel a navrhněte aplikaci, která umožní pohodlné sestavení kontrolního seznamu.
Implementujte aplikaci pro operační systém Android vhodnou pro telefony i pro tablety.
V aplikaci implementujte vhodné kategorie, omezení (váha), historii a pokuste se navrhnout vhodný způsob adaptace (doporučení).
Umožněte vhodný import a export dat a navrhněte proces sdílení seznamů mezi několika uživateli.
Výslednou aplikaci otestujte s uživateli a ověřte funkčnost i použitelnost (usability) uživatelského rozhraní.

Seznam odborné literatury

Dodá vedoucí práce.

Ing. Michal Valenta, Ph.D.
vedoucí katedry

doc. RNDr. Ing. Marcel Jiřina, Ph.D.
děkan

V Praze dne 10. ledna 2018

FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Bachelor's thesis

Travel Checklist for Android devices

Marek Alexa

Department of Software Engineering
Supervisor: Ing. Vratislav Zima

May 15, 2018

Acknowledgements

I would like to thank my supervisor, Ing. Vratislav Zima, for supervising this thesis. His advice and remarks were constructive. I would also like to thank my family and my friends for supporting me during my studies. Last but not least, I would like to thank everyone that has helped me with 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 accordance with Article 46(6) of the Act, I hereby grant a nonexclusive authorization (license) to utilize this thesis, including any and all computer programs incorporated therein or attached thereto and all corresponding documentation (hereinafter collectively referred to as the "Work"), to any and all persons that wish to utilize the Work. Such persons are entitled to use the Work in any way (including for-profit purposes) that does not detract from its value. This authorization is not limited in terms of time, location and quantity. However, all persons that makes use of the above license shall be obliged to grant a license at least in the same scope as defined above with respect to each and every work that is created (wholly or in part) based on the Work, by modifying the Work, by combining the Work with another work, by including the Work in a collection of works or by adapting the Work (including translation), and at the same time make available the source code of such work at least in a way and scope that are comparable to the way and scope in which the source code of the Work is made available.

In Prague on May 15, 2018 .....

Czech Technical University in Prague
Faculty of Information Technology

Citation of this thesis

Alexa, Marek. *Travel Checklist for Android devices.* Bachelor's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2018. Also available from: ⟨`https://github.com/skvaryk/android-travelcheck`⟩.

Abstrakt

Cílem této bakalářské práce bylo navrhnout a implementovat mobilní aplikaci pro OS Android, která umožní jejím uživatelům pohodlné sestavení cestovního seznamu. Byla provedena analýza současných řešení problematiky, návrh a implementace aplikace a následné otestování s uživateli testy použitelnosti.

Klíčová slova Android, mobilní aplikace, cestování, zavazadla, cestovní seznam

Abstract

The goal of this bachelor thesis was to design and implement a mobile application for OS Android, which will allow its users comfortable compilation of a travelling list. An analysis of current solutions of the problematics, solution proposition, and implementation of the application and subsequent usability testing were all done.

Keywords Android, mobile application, travelling, luggage, travel checklist

Contents

List of Figures

List of Tables

Introduction

Travelling is steadily rising in popularity year by year and so is the need to keep luggage organized. Because finding out that you have forgotten to pack a notebook adapter for a business trip presents a well-justified fear, people have been keeping lists of travelling necessities to avoid similar scenarios.

Handwritten lists, albeit familiar, have a number of disadvantages. Foremost, travellers have to put them together by themselves, and they are susceptible to physical damage or loss. It could be argued, that there are thought out lists for various occasions, however, those rarely fit one's needs.

The age of smartphones is upon us for a few years now. Therefore the solution is at hand – an application for smartphones that facilitates the creation of a travel checklist and adapts to individual needs of users. Android OS presents with 85% [3] market share one of the technological trends in the smartphone market.

The goal of this thesis is to create a user-friendly application for Android OS. The application should handle creation and management of travel checklists and serve as a modern substitution of a handwritten checklist. Specifically, this thesis consists of an analysis of present solutions, design, and implementation of a solution. The result will be tested by targeted users.

Chapter 1

Analysis

The goal of analytical phase is to collect enough information to be able to take care of user's needs. This chapter comprises an analysis of possible solutions and currently available solutions. Gathered information will crystallize in the form of a list containing both functional and non-functional requirements, which will be used when it comes to designing the application.

1.1 Problem definition

The problem this thesis is trying to solve is rather simple. The user has a need to keep a list of items they want to bring with them to a holiday or a trip. The difficulty comes with trying to optimize this process and make it as comfortable as possible for the user.

1.2 Possible solutions

1.2.1 Trivial solution

The problematics of a travel checklist could be solved very easily – pencil with a piece of paper is enough for the user to form a list. This solution, though simple and practical, is very time-consuming and the burden of actually picking out the items lies on the user.

Therefore, it would be a better idea to try and solve this problem with the help of electronic devices. Possible devices include a personal computer, mobile device, or even smart television, however mobile devices make the most sense, as they are widely spread among young people, the dominant target group. Another advantage is the accessibility of smartphones.

With mobile devices chosen as the platform for this solution, it is still needed to select the targeted operating system. The two dominant mobile operating systems are Android and iOS, which together have 99.9 % market share, as can be seen in table 1.1.

Operating system	2017 Market Share	2017 Units
Android	85.9 %	1,320,118
iOS	14.0 %	214,924
Other OS	0.1 %	1,493

Table 1.1: Mobile operating system market share

I have chosen Android OS because it is open-source, it is by far the most prominent mobile OS, and because it has low hardware prerequisities.

1.2.2 Trivial solution for Android platform

A trivial solution for OS Android would be an application that imitates the mentioned pen and paper solution. The user would be presented with a text field, where he could fill out an item designated to be packed, and confirm by pressing a button. Once approved, the item will be added to the list, where it can be checked or deleted.

In comparison to pen and paper, this solution has the advantage of always being at hand, but that is it. User comfort is comparable and time demands are higher when compared to writing your list by hand.

1.2.3 Improved solution for Android platform

Problems mentioned in the previous subsection could be alleviated by already having a list of items from which the user could choose.

Due to the large number of possible items to choose from, it would be sensible to organize them into categories. Organizing items into groups can be tackled from a few different perspectives. For example, elements can be divided into clothing, electronics, hygiene, etc. Another option could be to organize the items based on the type of landscape of the destination. However, the best solution was found to be to divide the items based on activities the user is planning to do on the trip – for example skiing, camping, business meeting, etc. It would also be wise to separate some items by gender.

1.2.4 Travel checklist generation

The problem that the user has to manually put together their checklist persists, which is agonizing and time-consuming. The solution is to generate the checklist by the application itself.

This solution can be implemented in a few ways. The easiest is to include all items from an activity in the checklist followed by manual modification. An improvement of this solution would be to add the influence of weather – an umbrella is not always necessary. Furthermore, it is possible to preserve item's

popularity in respect to a specific user and choose items from given activity based on the item's popularity. By implementing popularity bound to items, the user's trip history is also projected into the generation of a checklist.

A good solution would be to generate travel checklists based on global statistics and subsequently modify them based on user's preferences and weather. Unfortunately, no such figures are freely accessible by the public. A solution to this problem would be to collect enough data from the application to make such a statistic. However, that is a long shot.

1.3 Current solutions

Currently, there are several applications for Android platform which address the problematics of creating and managing travel checklist. However, only a few of them are worth analyzing, as most of them do not offer more than a simple checklist. Applications were mainly chosen based on rating and number of downloads. Specifically, PackMeApp [4], PackKing [5] and PackPoint [6] were chosen. From the table 1.2 we can see that all analyzed applications have had at least a mild success.

Application name	Rating	Number of downloads
PackMeApp	4,0	50 000+
PackKing	4,6	100 000+
PackPoint	4,6	500 000+

Table 1.2: Analysed applications summary based on [2]

1.3.1 PackMeApp

A straightforward user interface (see fig. 1.1) of the application PackMeApp [4] serves its function well enough and sticks to a uniform design. Navigating between individual screens can be confusing because the application does not follow Android guidelines [7].

The user creates their checklist with the help of in-app items, which are organized by categories. The application allows adding new items to existing categories but does not allow creating new categories. Items have a relevance rating, by which they are sorted.

1.3.2 PackKing

The user interface (see fig. 1.2) of the application PackKing [4] is comfortable and easy to use, the most noticeable flaws being missing translations and confusing transitions between screens.

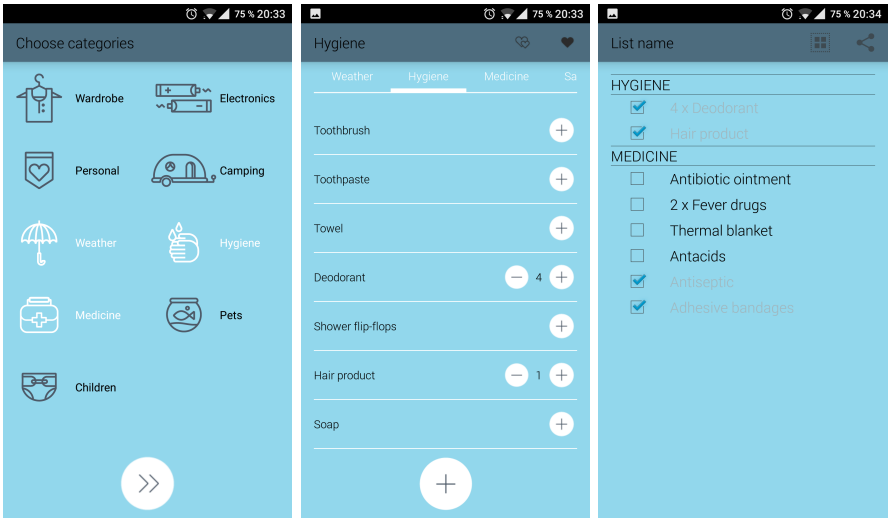

Figure 1.1: PackMeApp overview

The user creates their checklist by selecting from a list of items. Items by divided by activities, weather, and means of transport. The application allows the creation of custom activities. All categories are editable, meaning the user can add new or remove existing items.

The application has a premium version, which offers the creation of more than one checklist at a time, export of a checklist to pdf or uploading list's backup to Google Drive [8].

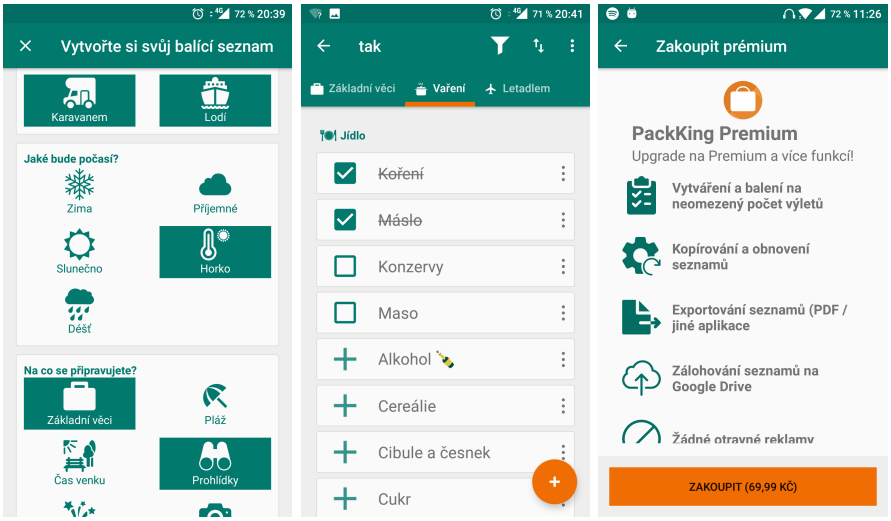

Figure 1.2: PackKing overview

1.3.3 PackPoint

The user interface (see fig. 1.3) of the PackPoint [6] application is organized, intuitive and practical. The application reacts adequately to user's gestures. The only controversial choice is frequent notifications.

Individual items are organized by activities and nothing else. The application differentiates between business and leisure activities. It seems like the application checks weather, but the weather itself doesn't project into checklist creation.

PackPoint [6] also offers a premium version, which allows the editing of item categories. Premium version also includes Tripit and Evernote integration.

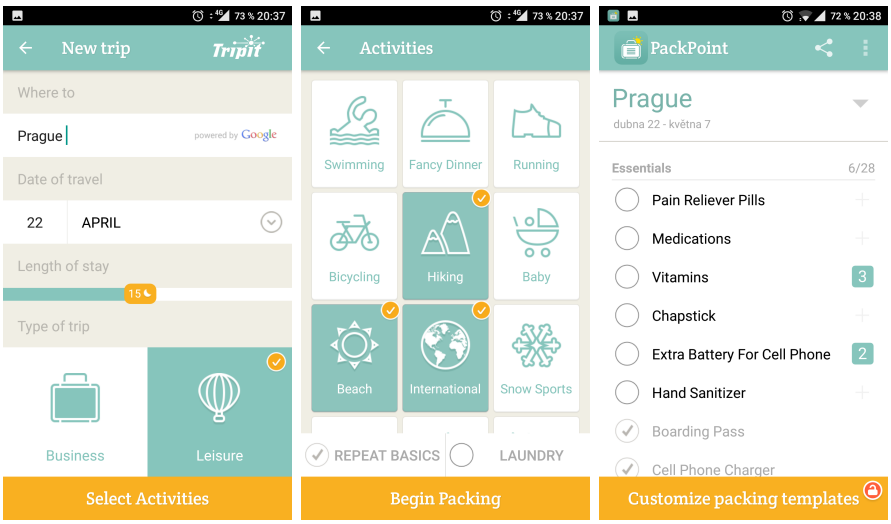

Figure 1.3: PackPoint overview

Chapter 2

Solution proposition

This chapter will be using findings from the previous chapter 1 to create a design that will lead to a successful implementation. The chapter comprises of chosen solution, functional and non-functional requirements, description of use-cases, UI (User Interface) design, and domain and database model.

All diagrams and models are modelled in Enterprise Architect [9] application. Enterprise Architect is a visual editor used for modelling and design based on the UML (Unified Modelling Language). It can be used to encompass full application development life-cycle.

2.1 Proposed solution

From the analyses of possible solutions (section 1.2) and current solutions (section 1.3) was drafted the following theoretical solution.

Items will be organized into activities, which will be used as templates for travel checklist generation. Items can be exclusive to one of the (two) genders or be unisex. The generating will also be influenced by weather – some items will be chosen only when it will snow, etc. Items will be attributed a score (popularity), based on which they will be decided whether to add an item to a list that is being generated. This highly adaptive solution was found to provide the best experience for the user.

The following table 2.1) compares proposed solution with analysed applications (see section 1.3. The table shows which feature is implemented by which application. From this comparison, the proposed solution stands out as the best.

Feature	PackMeApp	PackKing	PackPoint	Proposed solution
Editation of categories	×[1]	✓	×	✓
Item suggestion	×	✓	✓	✓
Weather influence	×	×	✓	✓
No advertisements	×	×	×	✓
User's gender influence	×	✓	✓	✓
List evaluation	×	×	×	✓
Export/import	×	×	×	✓

Table 2.1: Analysed applications and proposed solution feature summary

2.2 Requirements

The following lists of functional and non-functional requirements have been compiled based on previous section proposed solution 2.1.

2.2.1 Functional requirements

- Trip's location selection
- Picking date and length of stay
- Activities selection
- Checklist generation
- Checklist organization – adding and deleting items and activities from given checklist
- General management of activities and items they are containing
- Gesture recognition
- Import and export of chosen checklists

[1]In the PackMeApp application, it is possible to add items to categories, but it is not allowed to add custom categories.

2.2.2 Non-functional requirements

- Support of phones and tablets with Android OS 4.1.x and above (API 16)[2]
- Use of integrated relational database system SQLite
- Download of weather from web API
- Multiple users in one application's instance
- Offline mode
- List generation adaptation

API (application program interface): a set of protocols used by programmers to create applications for a specific operating system or to interface between the different modules of an application. [11]

SQLite [12] database is a software library that implements a database engine, which is self-sufficient and without a need for a server or any configuration.

List generation adaptation means collecting data about the user (mainly past trips), which will be used in new checklist compilation, while also taking weather into account.

2.3 Use cases

From requirements laid out in section 2.2 can be comfortably created use-case model (fig. 2.1). The model will be helpful in creating user interface and in formulating scenarios for usability tests.

The application will start on a list of created trips, wherefrom the user can choose any depicted action.

The action Geolocation selection serves mainly for weather forecasting, which will help in checklist generation. This action will not be accessible in offline mode.

The difference between adding item and categories in the View trip and Activities management actions is that when adding an item to a trip, the item is added to both the current checklist and a database for later usage in trip generation. Whereas adding items in Activities management will not add them to any trip's checklist. In the case of item deletion, the situation reverses. Deleting an item from a trip's checklist will not remove it from the database, but deleting an item from the database will delete it from all the trips.

[2]This represents 99.3 % of all Android devices [10]

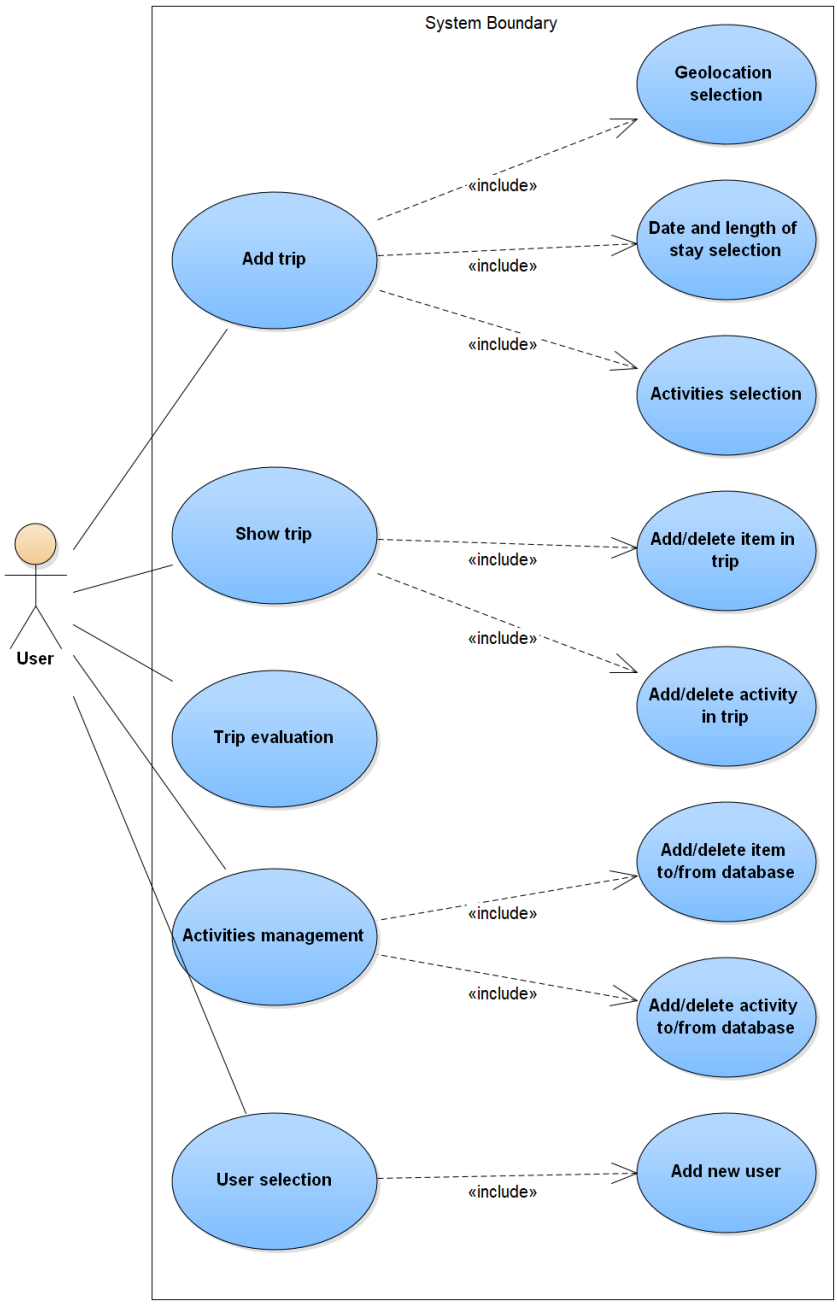

Figure 2.1: Use-case model

2.4 User interface design

The user interface is designed with an emphasis on intuitiveness and practicality. The online tool NinjaMock [13] was used in the design process to create mockups.

The design is based on use-cases, which were defined in section 2.3. This model provides information about the structure and functionality of the user interface. The design has undergone changes in the implementation phase.

2.4.1 Trip list

The GUI (graphical user interface) design (see fig. 2.2) of the Trip list screen contains, as the name suggests, a list of created trips. After clicking on a trip, one of two actions follows. Those two actions are Show trip or Trip evaluation (referencing use-case model 2.1). Which action takes place depends whether the trip has elapsed or not.

The design also includes a button on the bottom portion of the screen, which will start series of actions designated for new trip creation.

Last two main actions (User selection and Activities management) can be initiated by clicking on the Overflow menu [14]. These two actions are expected to be the least frequent, so they are relatively small and unobtrusive in the screen design.

Figure 2.2: Trip list - GUI design

2.4.2 Add trip action

The action Add trip comprises of three consecutive actions and does not exist on its own.

The user interface (see fig. 2.3) of the Geolocation selection action is a straightforward one. The user will be shown a map, where they will choose a location by tapping on the map. The selection has to be confirmed by a button on the bottom of the screen. The action Date and length of stay selection follows.

The interface (see fig. 2.3) of the Date and length of stay action is rather simple. The user is shown two text fields with labels, where they will enter the date of start and end of their trip. After confirming, the Select activities screen is displayed.

The GUI design (see fig. 2.3) of Select activities[3] contains a grid layout of activities' icons, which can be selected by clicking. After confirming by a button on the bottom of the screen, the Show trip action will follow.

Figure 2.3: Add trip - GUI design

2.4.3 Show trip

The design of this GUI (see fig 2.4) holds a list which can contain four different items: a category, an item in a category, and two buttons for adding new item or category. The list will support checking items.

[3]The activities are referencing categories of items, not Android activities.

Figure 2.4: Show trip - GUI design

2.4.4 Trip evaluation

The user interface of Trip evaluation has a similar design to the Show trip action's user interface (see fig. 2.4). However, it serves for selecting which items were useful and which were not, thus evaluating the trip's checklist. Unlike the Show trip design, this design does not contain buttons for adding new items or categories.

2.4.5 User selection and Add new user

These two actions are for practical reasons designed as pop-up windows.

The GUI (see fig. 2.5) of User selection contains a list of users and a button to add a new user, which facilitates the Add new user action.

The design of the action Add new user (not shown in the figure) comprises of a text field for the new user's username and a button used for gender selection.

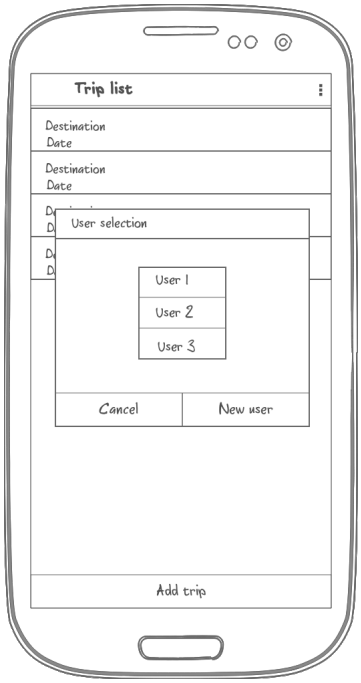

Figure 2.5: User selection – GUI design

2.4.6 Activities management

The user interface is of Activities management is based on the design of View trip design. However, the interface only serves as template management. Template management means adding and deleting items and categories from the database.

2.5 Domain model

A domain model is created together with the use-case model (see fig. 2.1) in the early stage of software development. A domain model is a form of a class diagram. The classes in domain model are simplified, as they do not contain methods and have only important attributes. A domain model is platform independent. [15, 16]

2.5.1 User

The entity User serves primarily to determine the ownership of a trip. It has two attributes representing user's name and gender.

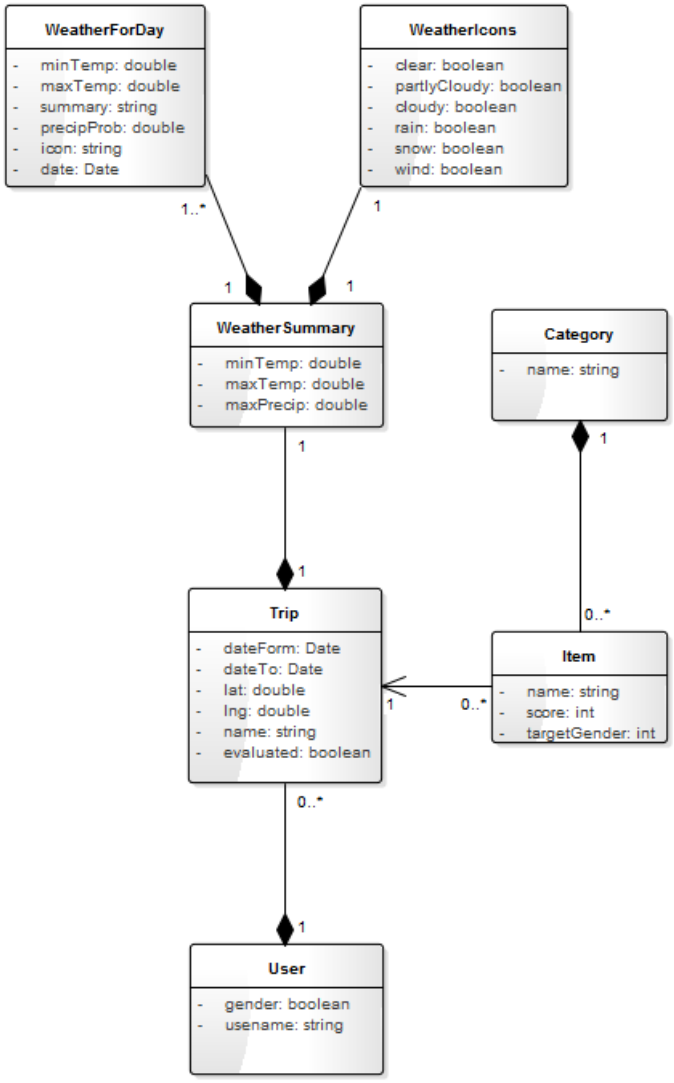

Figure 2.6: Domain model

2.5.2 Trip

Trip is the central entity of the domain model. Every trip contains a date of start and end, geolocation, trip's name and an indication, whether the trip has been evaluated. Furthermore, the entity contains one WeatherSummary and can contain any amount of Items.

2.5.3 WeatherSummary

The entity WeatherSummary contains a summary of the weather forecast during the stay. It comprises of lowest temperature, highest temperature, and the highest precipitation probability. In relation to other entities, WeatherSummary contains one WeatherIcons entity and several WeatherForDay entities, the exact number being the number of days of a planned stay.

2.5.4 WeatherForDay

The WeatherForDay entity represents weather forecast for one day. Its attributes are minimal temperature, maximal temperature, text summary, precipitation probability, a text representation of an icon and a date to which the forecast applies.

2.5.5 WeatherIcons

WeatherIcons is a collection of six indicators for weather icons. It is used for saving all possibilities of a weather forecast summary.

2.5.6 Category

The Category[4] represents a category of items and its only attribute is a name.

2.5.7 Item

The Item entity represents an item which can be packed for a trip. Its attributes are name, score and target gender. Items belong to a category.

2.6 Relational model

A relational model specifies the domain model usage for concrete database system (in this case SQLite). SQLite support only five data types [17]. These data types are:

- NULL – The value is a NULL value.
- INTEGER – The value is a signed integer, stored in 1, 2, 3, 4, 6, or 8 bytes depending on the magnitude of the value.
- REAL – The value is a floating point value, stored as an 8-byte IEEE floating point number.
- TEXT – The value is a text string, stored using the database encoding (UTF-8, UTF-16BE or UTF-16LE).

[4] In the context of a user interface, categories are called activities.

- BLOB – The value is a blob of data, stored exactly as it was input.

Because of the data types constraints, it is needed to adapt the data types from the domain model 2.6. The only three relevant data types are integer, real, and text. For example, a date is converted into text.

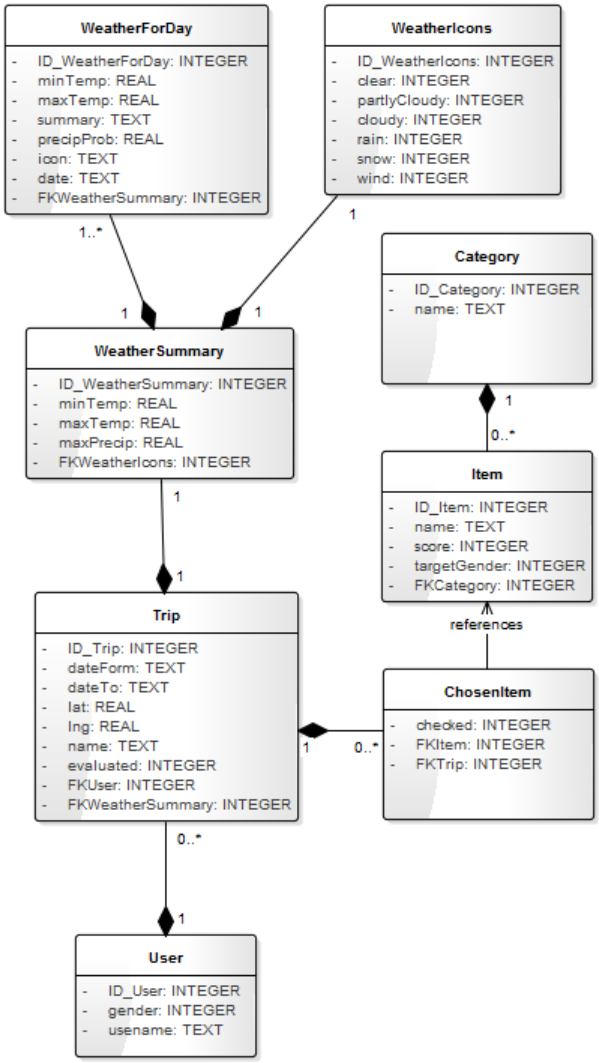

Figure 2.7: Relational model

Primary keys in this relational model are marked with ID (identifier), while foreign keys have FK (foreign key) prefix. The keys distribution is based on the cardinality of relations. The relation between Trip and Item (M:N cardinality) had to be resolved by adding another table named ChosenItem, which contains

two foreign keys and no primary key – an entry is identified by combining these two foreign keys into a composite key. The M:1 relations is resolved by adding a foreign key attribute to the table with M cardinality. In the 1:1 cardinality case, it does not matter where the foreign key is put.

Chapter 3

Implementation

For the development phase, mainly Android Studio version 3.1.2 [18], including an integrated LayoutEditor [19]. The code is written in Kotlin language [20] version 1.2.0. The minimum SDK version is 14 (Android 4.0) because lower SDK versions do not support newest Android support libraries [21]. The targeted SDK is 27, which corresponds to Android 8.1. A Gradle system [22] is used for project compiling. For persistence, the SQLite [12] database system is used in version 3.7.4+ (depends on Android OS version).

Kotlin is a statically typed programming language for modern multiplatform applications that runs on the Java virtual machine. Since Android Studio 3.0, Kotlin is fully supported programming language on the Android OS.

SDK (software development kit) is a programming package that enables a programmer to develop applications for a specific platform. Typically an SDK includes one or more APIs, programming tools, and documentation. [23]

3.1 Used libraries

For the implementation, two external libraries were used. First library is Gson (more in detail in subsection 3.1.1). The second external library is Google Play Services [24], which facilitates the usage of Google maps in the application. Furthermore, three Android support libraries were used, mainly to lower the minimum SDK requirement.

3.1.1 Gson

Gson is a Java library that can be used to convert Java Objects into their JSON representation. It can also be used to convert a JSON string to an equivalent Java object. Gson can work with arbitrary Java objects including pre-existing objects that you do not have source-code of. [25]

3.1.2 Android support libraries

Android support libraries mainly offer backward compatibility of features, classes, and methods implemented in newer SDKs. They can also provide convenience and helper classes, and utilities (for example support for annotations). [26]

List of used support libraries:

- **v7 appcompat library:** For backwards compability of ActionBar [27].
- **v7 gridlayout library:** Because the application uses GridLayout class [28].
- **Multidex Support Library:** Needed for applications that exceed 64K reference limit. [29]

A list of all support libraries and their description can be found in official documentation. [26].

3.2 Database implementation

As was mentioned in the previous section, the chosen database engine is SQLite, which is integrated into Android and has direct access to a database. All requests for getting, deleting or saving data are facilitated by DatabaseManager class (see code example 3.1). DatabaseManager has a static variable instance, which is set when the application starts. Thanks to this static variable, the DatabaseManager is accessible from anywhere in the application.

For initialization, opening and general administration of the database, the MySQLiteHelper class is used. MySQLiteHelper inherits from abstract class SQLiteOpenHelper [30] and contains scripts for creating tables according to the relational model (section 2.6). It also includes scripts for initial database fill with basic data (items and categories).

Listing 3.1: Example of acquiring an entry from the Item table in the DatabaseManager class.

```
public Item getItemByID(int itemID) {
    Cursor cursor = db.query("item", new String[]{
        "ID_item","name", "score", "targetGender",
        "FKCategory"},
        "ID_item = " + itemID, null, null, null,
         null, null);
    if (cursor.moveToFirst()) {
        Item item = new Item();
        item.setID(cursor.getInt(0));
        item.setName(cursor.getString(1));
```

```
            item.setScore(cursor.getInt(2));
            item.setTargetGender(cursor.getInt(3));
            item.setFKCategory(cursor.getInt(4));
            cursor.close();
            return item;
        } else {
            cursor.close();
            return null;
        }
}
```

3.3 Web API

In the application, web API with the name The Dark Sky Forecast API [31] was used for getting a weather forecast. The authors offer a thousand calls for forecast a day without a fee. This amount is sufficient for developing an application with a small userbase. However, if the application will get larger, a change of API will be required. The other option is to start paying for the usage. An example of calling the API:

```
https://api.forecast.io/forecast/*key*/
50.43884235565519,14.882826134562492,1467151200?
units=si&exclude=hourly,currently
```

In place of the part of address marked as *key* should be private key to the web API.

Following are three numbers separated by a comma. These three numbers are latitude, longitude and Unix timestamp respectively. The Unix time (or Unix epoch or POSIX time or Unix timestamp) is a system for describing points in time, defined as the number of seconds elapsed since midnight proleptic Coordinated Universal Time (UTC) of January 1, 1970, not counting leap seconds. [32]

The first of the two parameters for this request specifies that the answer should be using the international system of units form. The second parameter specifies that we do not need a forecast for individual hours and a current forecast. For more information about this API, please see online documentation [33].

The answer has a JSON format, containing weather forecast for one day. The response string is then processed with the help of the Gson library (see subsection Gson 3.1.1).

3.4 Import and export of trips

The export of a trip begins by invalidating a Trip's ID. Then follows a serialization of TripExportContainer, which contains a Trip, an array of Items, and an array of corresponding Categories. The serialization is done through Gson library (see subsection Gson 3.1.1). The resulting string is written into a file.

The import of a trip can be initiated by clicking on file with json extension, as the application is registered to react to this intent. By deserializing the exported file, we get a trip object. However, it is still needed to set the trip's ID, user's ID, and to get the weather forecast. The resulting trip is saved into the database.

3.5 Activity design

Activities are one of the fundamental building blocks of apps on the Android platform. They serve as the entry point for a user's interaction with an app, and are also central to how a user navigates within an app (as with the Back button) or between apps (as with the Recents button). [34]

A software framework is a concrete or conceptual platform where common code with generic functionality can be selectively specialized or overridden by developers or users. Frameworks take the form of libraries, where a well-defined application program interface (API) is reusable anywhere within the software under development. [35]

In simplified terms, activity is one screen, with which the user can interact. Because of this, the activity model is crucial for designing the user interface. The activity model defines relations between individual activities. The diagram 3.1 shows an activity lifecycle and its behavior under varied circumstances.

3.5.1 Activity model

The main activity is the Trip list activity. This activity contains a list of created trips and serves as the starting point for all other actions. The activities User selection and New user creation are designed as Dialogs, which are more appropriate for these fast and simple actions. Conversely, the activity Add trip is broken down into three sequential activities. The Add trip activity does not exist on its own. From any activity, it is possible to navigate in the opposite direction of an arrow by clicking on the back button. The only exception to this back navigation is Show trip activity, from which the user cannot return to Add trip.

A Dialog[5] 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

[5] From now on referred to as a dialog.

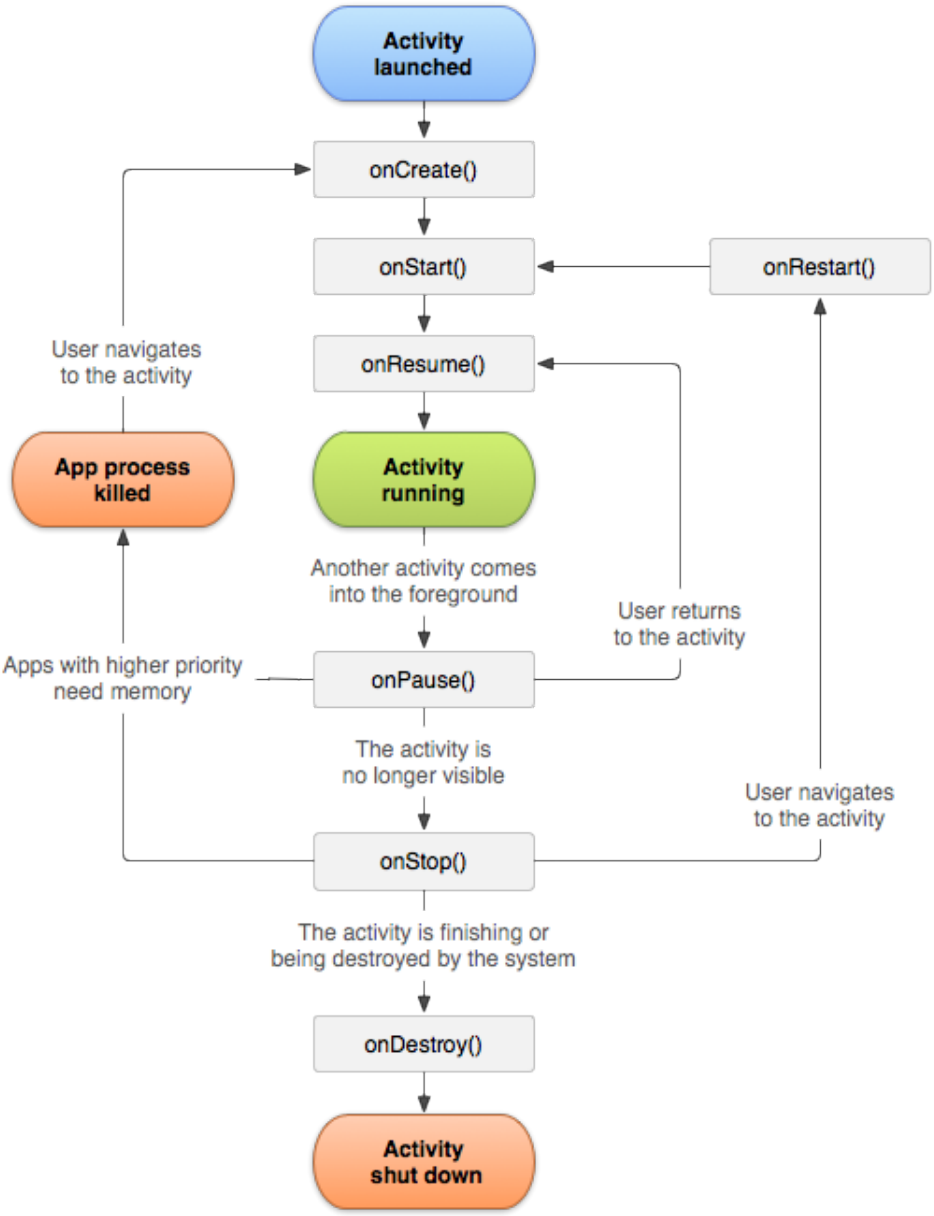

Figure 3.1: Activity lifecycle [1]

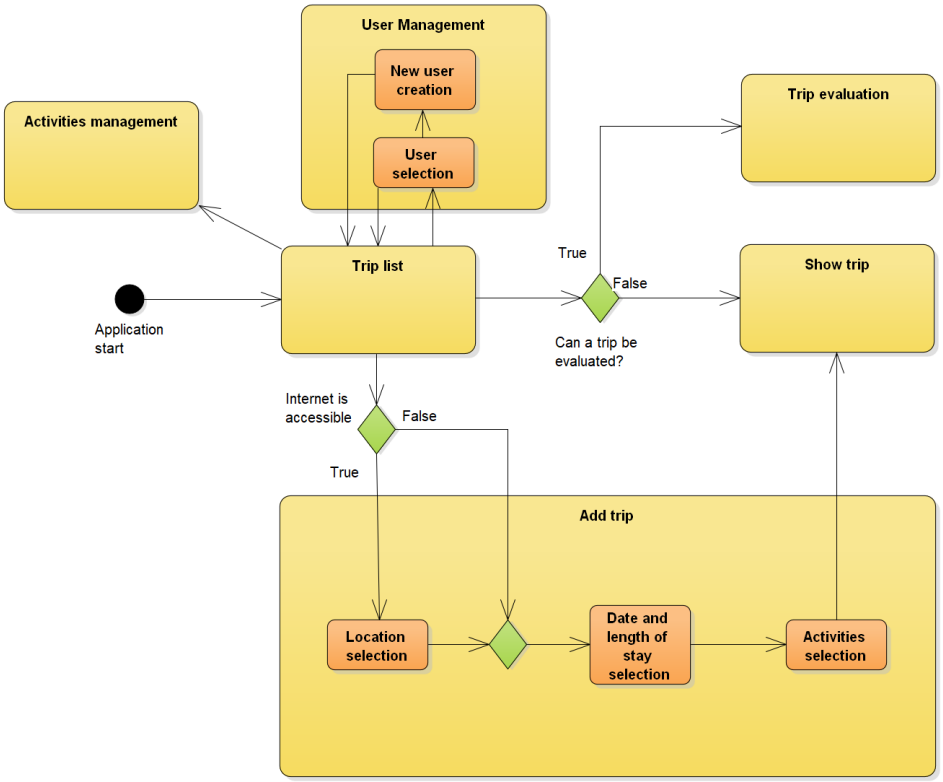

Figure 3.2: Activity model[6]

used for modal events that require users to take an action before they can proceed. [36]

3.6 Activity implementation

The activity implementation is based on Activity design (section 3.5) and User interface design (section 2.4). The functionalities of the implemented activities are described in section User interface design.

The user interface is designed for a vertical screen layout. However, thanks to the implementation of ScrollView [37], the application can handle even horizontal screen layout.

All activities (not dialogs) use ActionBar [27]. ActionBar's creation is handled by ToolbarFactory class. ToolbarFactory uses an android support library to create toolbars when requested through a static method createToolbar. The

appearance is uniformly defined in the file `toolbar.xml`.

The activities Trip list and all the activities that are realizing Add trip use a uniform button on the bottom of the screen. The button is defined in `custom_button`.

Several adapters were used in the implementation. These adapters define the appearance and reactions on user's actions for individual items in ListView [38]. All the adapters inherit from BaseAdapter [39] class.

3.6.1 Trip list

The Trip list activity is the main activity, which means, that the application starts with this activity. If the application is run for the first time, a dialog for New user creating is shown to create the first user. The active user ID is saved to shared preferences (see SharedPreferences [40]). If none is found in the preferences, the first created user is marked as the active user.

A trip that has passed is distinguished by having its name crossed out. Furthermore, trips that have been evaluated are marked with a green checkmark. A trip can be deleted by swiping gesture.

A ListView [38] is responsible for layout and deciding how to react to user's actions. The ListView uses TripListAdapter as an adapter.

The resulting implementation of the user interface design (subsection 2.4.1) can be see in figure 3.3.

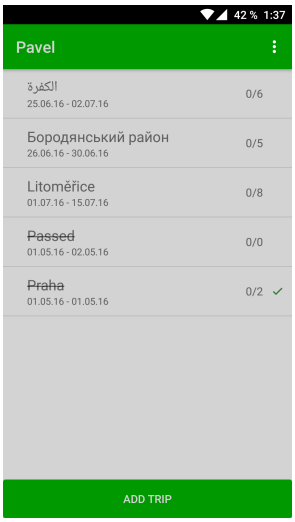

Figure 3.3: Trip list implementation

[6] Activities selection and Activities management in this model do not refer to Android activities, but to a category of items.

3.6.2 Add trip

As was meantioned earlier in section 3.5.1, the activity Add trip consist of three subsequent activities and does not exist on its own.

The first activity Location selection only displays a map and a button for confirmation. The map is implemented through a Google Play Services [24] library. The map itself is represented by GoogleMap [41] class. The user selects a location by clicking on the map and confirming by the button on the bottom of the screen. The confirmation button is inactive (greyed out) until the user clicks the map. After clicking on the confirmation button, the activity Date and length of stay selection is initiated, which is passed on the geolocation and trip name (from address) information by using the Intent [42] class.

The following activity is Date and length of stay selection. The user chooses a date by clicking on a CalendarView [43]. For displaying the length of stay, two TextViews [44] are used that mark the beginning and end of a trip. Which textview is currently being changed is indicated by a green frame around the textview, which displays when the view has a focus. The focus is switched between the two textview after every date selection. It is also possible to change to focus by tapping the textview itself. The confirmation button is inactive if the start of the trip is in the past, or if the end of the trip is before the start of the trip. The number of possible days a trip can have is bounded above by thirty days. After confirming the selected dates, an Activities selection activity is started, which is sent information from previous activity (geolocation and a name) and the start and end dates of the trip.

Before starting the Choose activities[7] activity, it is needed to download the weather forecast with the help of a web API (see section 3.3). Acquiring the forecast can take up to a few tens of seconds, depending on the number of days. Because of this delay, a ProgressDialog [45] is shown with information on which day's forecast is being downloaded. The acquired data are processed and displayed in a weather summary. The activity comprises a weather summary and a GridLayout [28] which contains all available categories of items in the form of pictures. For displaying these categories, a custom CheckableImageView class is used. The CheckableImageView class inherits from ImageView [46] class and add the ability to select or unselect a picture by applying a gray filter over the image. The same custom class is used for weather icons, in the case the user decides they do not want the weather to affect the list generation. After confirming the selection, all information about the trip is stored in a database and Show trip activity is initiated.

The implementation (see fig. 3.4) of user interface differs from the design (subsection 2.4.1) in two notable ways. The Date and length of stay activity uses a CalendarView to select a date instead of two text fields. The second difference is the inclusion of a weather summary in Choose activities screen.

[7] Choose activities refers to categories of items, not Android activities.

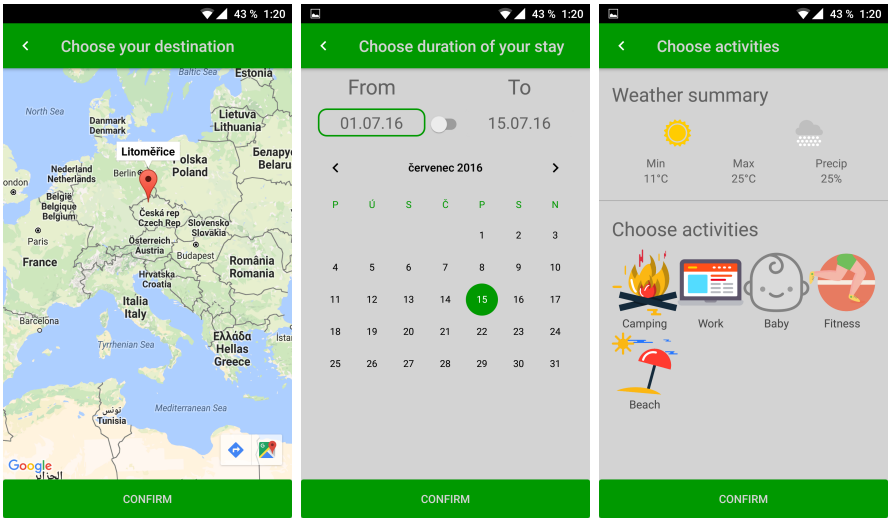

Figure 3.4: Add trip implementation

3.6.3 Show trip

The Show trip activity consists of two main components: a weather summary at the top and a Listview, which has its adapter – ViewTripAdapter. This adapter distinguishes between three types of items: a category title, an item in a category, and buttons to add new or existing items into a category. The categories can be added by clicking on the Overflow menu [14] and choosing the appropriate action. The categories are sorted alphabetically. Items are sorted by being checked or not and secondary by alphabetical order. A category will collapse when clicked, hiding items it contains. A user can check or uncheck an item by clicking on it. The list reacts on the horizontal swipe gesture by removing the swiped item. If the last item is deleted from a category, the category will be automatically deleted too. The buttons for adding new or existing items are at the end of each category.

After clicking on Add existing item button, a new dialog will be displayed with a list of all the items from given category. By clicking on Add new item button, a different dialog will be shown. This dialog has an EditText field for the new item's name, and a Spinner [47] for specifying the item's targeted gender. The dialogs for adding categories (accessed through the overflow menu) are almost the same. The only difference is that you cannot choose a targeted gender a new category.

The differences between user interface implementation (fig. 3.5) and the design (subsection 2.4.3) are adding a weather summary and moving the add category buttons to overflow menu. The weather summary is identical with the one displayed in Select activities implementation.

The overflow menu also contains a few new features. It is now possible to

set the trip's name, and if the trip was created without access to the internet, a user could specify its location, allowing it to receive weather forecast.

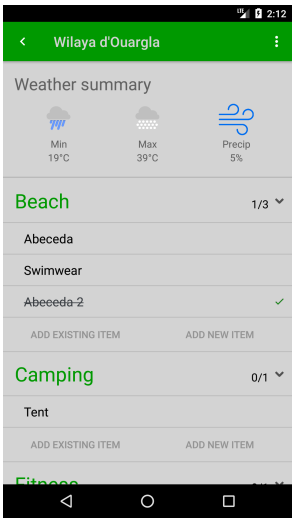

Figure 3.5: Show trip implementation

3.6.4 Trip evaluation

The Trip evaluation activity is very similar to the Show trip activity. It also has a ListView with its adapter, but it has far fewer functionalities. The only one left over is the ability to check items. However, the items are checked with a different icon. By going back from this activity, the displayed items have their score adjusted in the database based on being checked out or not (checked items are the ones the user did not need).

As the user interface design has suggested (subsection 2.4.4), the implementation is based on the Show trip activity (subsection 3.6.3). The resulting implementation can be seen in figure 3.6.

3.6.5 User selection and New user creation

The activities User selection and New user creation are implemented as dialogs. The dialog User selection contains a ListView with its adapter, the UserListAdapter class. More information about these dialogs is accessible in the Solution proposal chapter (subsection 2.4.5). Following a User selection action, the title in the action bar will be changed to their name and their trips will be shown. The implementation of these dialogs can be seen in figure 3.7.

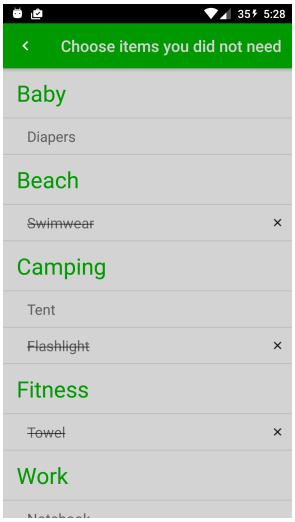

Figure 3.6: Trip evalution implementation

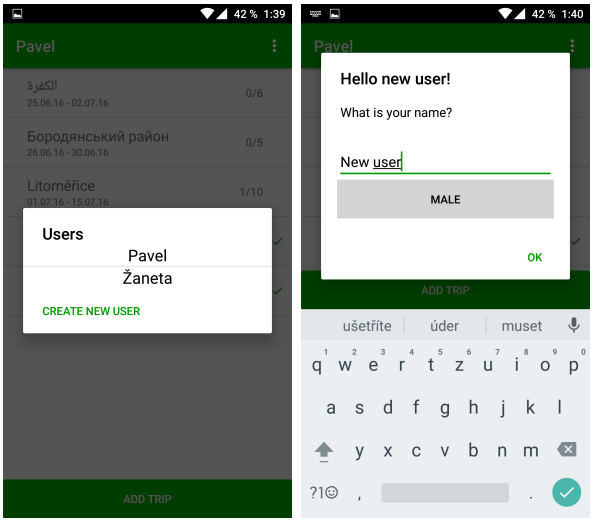

Figure 3.7: User selection and New user creation implementation

3.6.6 Activities management

The Activities management activity only has a ListView, which uses ManageActivitiesAdapter class as an adapter. After clicking on an item in the list, a dialog that asks the user to confirm deletion is shown. Either items or whole categories can be deleted. The creation of new items or categories is the same as in Show trip activity. Changes made to this list will be reflected in the templates in the database. This activity is accessible by clicking on the overflow menu in Trip list activity. The implemented user interface can be

seen in figure 3.8.

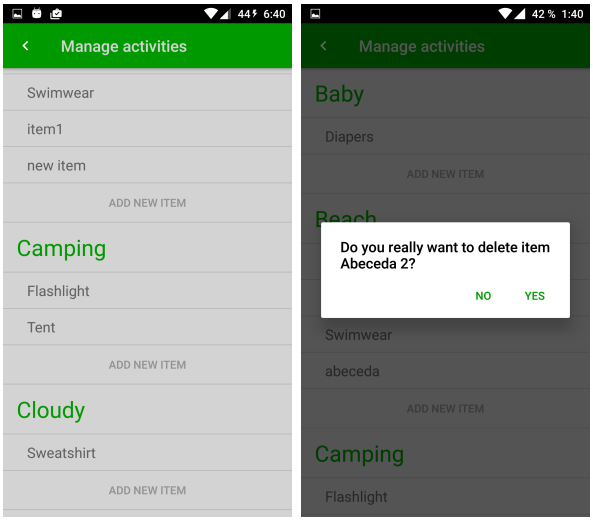

Figure 3.8: Activities management implementation

3.7 Adaptivity

The term adaptation in computer science refers to a process, in which an interactive system (adaptive system) adapts its behavior to individual users based on information acquired about its user(s) and its environment. Adaptivity indicates a system that adapts automatically to its users according to changing conditions, i.e., an adaptive system. [48]

A weather forecast in the context of the developed application is information about the environment. The information about the user is their gender, trip evaluation and chosen items in trips. The behaviors adaptation manifests in the travel checklist generation, which is affected by the collected information. This adaptivity system was determined based on the section Chosen solution 2.1.

Chapter 4

Testing

Given that the application's success is dependent on a simple and practical GUI, the main part of tests is usability testing with users.

Besides the usability testing with users, the application has been tested on these devices and emulators:

- Oneplus 2 (Android 6.0.1)
- Sony Bravia KD-55XD8005 (Android 6.0)
- Sony Xperia Z2 (Android 5.1.1)
- Nexus 5X (Android 4.4.2)
- Emulated Nexus 10 (Android 6.0)
- HUAWEI VNS-L21 (Android 7.0)

4.1 Usability testing

Usability testing refers to evaluating a product or service by testing it with representative users. Typically, during a test, participants will try to complete typical tasks while observers watch, listen and takes notes. The goal is to identify any usability problems, collect qualitative and quantitative data and determine the participant's satisfaction with the product. [49]

The application was tested by six users on their devices. Some notes and remarks were personally written, that is why some answers are just comments on the user's behavior. The first and second respondent have experience with application development, while other users are ordinary users. The users were given a modified version of the application with test data in order for the user to be able to test all features. The users were asked to be critical before the test started. Following are the results of the testing in the form of a question (or a task) and a list of answers from the individual users.

1. Create new user:
    - I'm missing an option to delete a user.
    - No problem.
    - A brief confusion, if the displayed value on ToggleButton corresponds with the chosen value.
    - Easy, there is nothing to be debated.
    - I'm missing a button to cancel the dialog.
    - Easy.
2. Add new trip:
    - Everything alright.
    - The application had stopped working on the first try when I tried to set the trip's date a year in advance. The second try with a sooner date was ok.
    - The user appreciates the information displayed in ProgressDialog when the forecast is being acquired.
    - The date choosing is a bit confusing. Personally, I would be satisfied with two text fields. The weather summary could be divided into days. Otherwise ok.
    - Too few activities to choose from.
    - Nice icons.
3. When viewing the trip, add an activity and add an item to the activity:
    - I would replace Fitness activity with Sports, and I would add a selection which sports the user plans to do. Based on this, I would choose more specific items that the user needs. I would also add an activity for a business trip, something where a person needs formal clothing and so on. But that could be put into the work activity.
    - Absolutely perfect, but it could be shown in a tutorial.
    - Very nice design. I love it.
    - No problem.
    - It works.
    - The user could not find the requested actions at first.

4. Delete items in a category until it disappears:
    - I would for sure leave in the option to delete a whole category.
    - I want to delete a category, maybe with a confirmation dialog.
    - There is no help to find out how to do this.
    - It took me a while to figure out how to delete an item.
    - The sensitivity for the deletion is too much. The gesture can be only indicated, and the item is deleted.
    - The user could not figure out how to delete an item. There was no problem after a small suggestion.
5. Delete an arbitrary item and category in the Activities management:
    - All right.
    - This could work the same as when viewing a trip. Or the other way around, in trips, it could function the same as here.
    - Intuitive, without a problem.
    - Functional.
    - No problem.
6. Switch to the user Pavel[8] and evaluate the trip named Praha:
    - OK.
    - A confirmation dialog really should pop-up here.
    - I have left the evaluation by mistake and it is now locked.
    - I have removed that trip by mistake.
    - Evaluation works great.
    - The user did not recognize that they are evaluating the trip.
7. General comments and reactions:
    - The gesture for deleting is too sensitive, resulting in unwanted item deletion when clicking.
    - No general comments.
    - I have described everything in previous questions.
    - The graphical interface is good, but the controls are not really intuitive. A tutorial would be nice.
    - Most things are serviceable, but the application could definitely take a few adjustments.
    - The application looks good. For me, as a layman, a few things were confusing, but next time I would know what to do.

[8]This is part of the testing data.

8. Score the application on a scale 1-5:

	Graphical interface	Intuitivity	Features	Overall
User 1	1	3	2	2
User 2	2	2	2	2
User 3	1	1	1	1
User 4	2	2	3	2
User 5	3	1	2	2
User 6	1	3	1	2

Table 4.1: Evaluation by users

9. Used phone or tablet.

   - Lenovo Vibe S1 (Android 5.0)
   - Oneplus One (Android 5.0.2: Hydroxen OS)
   - Lenovo S90 (Android 4.4)
   - Samsung Galaxy S4 mini (Android 4.4.4)
   - Lenovo Vibe Shot (Android 5.1)
   - Aligator S5500 Duo HD IPS (Android 4.4.2)

4.2 Conclusion from testing

The comments from users that have tested the application have resulted in these changes in the application:

- An option to delete a user has been added.
- Negative buttons have been added to dialogs.
- Acquiring the feather forecast a long time in advance has been fixed.
- An option to delete a whole category was added.
- The deletion gesture's sensitivity has been lowered.
- A dialog has been added when leaving the trip's evaluation

I would like to mention again that the data the user have been given were only for testing purposes. From the user's comments and answers, a need for a tutorial has emerged. The tutorial has not been implemented due to time constraints.

Conclusion

The goal of this thesis was to analyze the needs of travelers when packing and subsequently, to design and implement a mobile application for Android OS. This application should help its users to put together and manage travel checklists comfortably. It should target smartphones and tablets. The application should have implemented these features: appropriate categories, limitations (weight), history, suitable means of adaptation (recommendations), import and export of data. Also, a sharing process of checklists between users should have been designed. The resulting application should have been tested with users, and the GUI's functionality and usability should have been verified.

The analysis of traveler's needs has been done through an analysis of current solutions (section 1.3). The result of this thesis is a mobile application for Android OS, with which a comfortable and effective assembly of a checklist is facilitated. The application is suitable for smartphones and tablets with vertical or horizontal screen layout. Categories (activities), trip history, import and export of checklists, and adaptation in the form of generation of a personalized checklist are implemented. However, limitations have not been implemented. The resulting application has been tested, mainly in the form of usability tests with users (section 4.1). From these tests, several minor changes have been revealed and fixed. The users were asked to rate the application at the end of the test. The following table 4.2 summarizes the results.

	Graphical interface	Intuitivity	Features	Overall
Average score	1,6	2	1,8	1,8

Table 4.2: Rating by the users in 1-5 scale

The application could be expanded upon in a few ways. The user's comfort can be improved by adding a tutorial in order to familiarize the user with the application. Internally, the application could be improved by implementing a modern database system, replacing the dated SQLite. The application could

be improved by implementing online shared checklists, but this would require a server for communication between individual applications. A server could also be used to collect data about the user's behavior. These data could be used to refine the templates from which the checklists are generated and for further improvement of the user experience.

Bibliography

[1] Android. Understand the Activity Lifecycle. [online], [Accessed: 13.5.2018]. Available from: https://developer.android.com/guide/components/activities/activity-lifecycle

[2] Google. Google Play Store. [online], [Accessed: 7.5.2018]. Available from: https://play.google.com/store

[3] Gartner, I. Gartner Says Worldwide Sales of Smartphones Recorded First Ever Decline During the Fourth Quarter of 2017. [online], [Accessed: 12.5.2018]. Available from: https://www.gartner.com/newsroom/id/3859963

[4] PackMeApp. PackMeApp Packing List. [mobile application], [Accessed: 7.5.2018]. Available from: https://play.google.com/store/apps/details?id=net.henrykratajczak.packmeapp

[5] Wer, M. Packing List for Travel - PackKing. [mobile application], [Accessed: 7.5.2018]. Available from: https://play.google.com/store/apps/details?id=com.adotis.packking

[6] Wawwo. PackPoint travel packing list. [mobile application], [Accessed: 7.5.2019]. Available from: https://play.google.com/store/apps/details?id=com.YRH.PackPoint

[7] Android. Designing Back and Up navigation. [online], [Accessed: 12.5.2018]. Available from: https://developer.android.com/training/design-navigation/ancestral-temporal

[8] Google. Google Drive. [online], [Accessed: 12.5.2018]. Available from: https://drive.google.com/

[9] systems, S. Enterprise Architect. [online], [Accessed: 25.6.2016]. Available from: http://sparxsystems.com.au/products/ea/index.html

[10] Google. Distribution dashboard. [online], [Accessed: 7.5.2018]. Available from: https://developer.android.com/about/dashboards/

[11] Dictionary. API. [online], [Accessed: 7.5.2018]. Available from: http://www.dictionary.com/browse/api

[12] SQLite. SQLite. [online], [Accessed: 11.5.2018]. Available from: https://sqlite.org/

[13] ninjamock.com. NinjaMock. [online], [Accessed: 10.5.2018]. Available from: https://ninjamock.com/

[14] Android. Menus. [online], [Accessed: 10.5.2018]. Available from: https://developer.android.com/guide/topics/ui/menus

[15] Mlejnek, J. Analýza problémové domény. [online], [Accessed: 11.5.2018]. Available from: https://edux.fit.cvut.cz/oppa/BI-SI1/prednasky/BI-SI1-P04m.pdf

[16] Scaled Agile, I. Domain Modeling. [online], [Accessed: 11.5.2018]. Available from: https://www.scaledagileframework.com/domain-modeling/

[17] SQLite. Datatypes In SQLite Version 3. [online], [Accessed: 26.6.2016]. Available from: https://www.sqlite.org/datatype3.html

[18] Android. Android Studio. [online], [Accessed: 11.5.2018]. Available from: https://developer.android.com/studio/

[19] Android. Design a UI with Layout Editor. [online], [Accessed: 11.5.2018]. Available from: https://developer.android.com/studio/write/layout-editor

[20] JetBrains. Kotlin. [online], [Accessed: 12.5.2018]. Available from: https://kotlinlang.org/

[21] Android. Support Library. [online], [Accessed: 12.5.2018]. Available from: https://developer.android.com/topic/libraries/support-library/#api-versions

[22] Gradle. Gradle. [online], [Accessed: 12.5.2018]. Available from: https://gradle.org/

[23] Beal, V. SDK - software development kit. [online], [Accessed: 12.5.2018]. Available from: https://www.webopedia.com/TERM/S/SDK.html

[24] Android. Google Play Services. [online], [Accessed: 12.5.2018]. Available from: https://developers.google.com/android/guides/overview

[25] Google. Gson. [online], [Accessed: 12.5.2018]. Available from: https://github.com/google/gson

[26] Android. Support Library. [online], [Accessed: 12.5.2018]. Available from: https://developer.android.com/topic/libraries/support-library/packages

[27] Android. ActionBar. [online], [Accessed: 12.5.2018]. Available from: https://developer.android.com/reference/android/app/ActionBar

[28] Android. GridLayout. [online], [Accessed: 12.5.2018]. Available from: https://developer.android.com/reference/android/support/v7/widget/GridLayout

[29] Android. Enable Multidex. [online], [Accessed: 12.5.2018]. Available from: https://developer.android.com/studio/build/multidex

[30] Android. SQLiteOpenHelper. [online], [Accessed: 12.5.2018]. Available from: https://developer.android.com/reference/android/database/sqlite/SQLiteOpenHelper

[31] Company, T. D. S. The Dark Sky Forecast API. [online], [Accessed: 12.5.2018]. Available from: https://developer.forecast.io/

[32] Ltd, V. About UNIX timestamp. [online], [Accessed: 12.5.2018]. Available from: http://unixtimestamp.50x.eu/about.php

[33] Company, T. D. S. v2 Forecast API. [online], [Accessed: 12.5.2018]. Available from: https://darksky.net/dev/docs

[34] Android. Activities. [online], [Accessed: 13.5.2018]. Available from: https://developer.android.com/guide/components/activities/

[35] Techopedia. Software Framework. [online], [Accessed: 13.5.2018]. Available from: https://www.techopedia.com/definition/14384/software-framework

[36] Android. Dialog. [online], [Accessed: 13.5.2018]. Available from: https://developer.android.com/guide/topics/ui/dialogs

[37] Android. ScrollView. [online], [Accessed: 13.5.2018]. Available from: https://developer.android.com/reference/android/widget/ScrollView

[38] Android. ListView. [online], [Accessed: 13.5.2018]. Available from: https://developer.android.com/reference/android/widget/ListView

[39] Android. BaseAdapter. [online], [Accessed: 13.5.2018]. Available from: https://developer.android.com/reference/android/widget/BaseAdapter

[40] Android. SharedPreferences. [online], [Accessed: 29.6.2016]. Available from: https://developer.android.com/reference/android/content/SharedPreferences

[41] Google. GoogleMap. [online], [Accessed: 29.6.2016]. Available from: https://developers.google.com/android/reference/com/google/android/gms/maps/GoogleMap

[42] Android. Intent. [online], [Accessed: 29.6.2016]. Available from: https://developer.android.com/reference/android/content/Intent.html

[43] Android. CalendarView. [online], [Accessed: 13.5.2018]. Available from: https://developer.android.com/reference/android/widget/CalendarView

[44] Android. TextView. [online], [Accessed: 13.5.2018]. Available from: https://developer.android.com/reference/android/widget/TextView

[45] Android. ProgressDialog. [online], [Accessed: 14.5.2018]. Available from: https://developer.android.com/reference/android/app/ProgressDialog

[46] Android. ImageView. [online], [Accessed: 14.5.2018]. Available from: https://developer.android.com/reference/android/widget/ImageView

[47] Android. Spinner. [online], [Accessed: 25.6.2016]. Available from: https://developer.android.com/reference/android/widget/Spinner.html

[48] Brogi A., P. E., Canal C. On the semantics of software adaptation. [online], [Accessed: 14.5.2018]. Available from: http://www.sciencedirect.com/science/article/pii/S0167642306000220

[49] usability.gov. Usability Testing. [online], [Accessed: 14.5.2018]. Available from: https://www.usability.gov/how-to-and-tools/methods/usability-testing.html

Appendix A

List of used abbreviations

OS Operating system

GUI Graphical user interface

UML Unified Modeling Language

API Application program interface

SDK Software development kit

JSON JavaScript Object Notation

UTF Unicode Transformation Format

Appendix B

User's manual

B.1 Requirements

The application is not graphically or performance demanding. The only limitation is Android OS version. The application supports Android version 4.1.x and above.

B.2 Application instalation

In order to install the application, it is needed to do these three following steps:

- Transfer the file TravelCheck.apk from the apk directory on the provided CD (or visit the GitHub page) into your devices.
- Find the transferred file with a file manager.
- Click on the found file to initiate the installation. (Make sure that you have allowed installation from unknown sources in the settings.)

B.3 Navigation in the application

The main screen is a trip list. Here you can find a list of future, current, and past trips. A new trip can be added by clicking on the Add trip button and subsequent selection of destination, date, and activities. After clicking on the overflow menu located in the top right, you can manage the activities (saved templates from which the checklists are generated) or change the user.

Appendix C

Programmer's manual

The project is using libraries described in section 3.1. The libraries are included in the provided project.

C.1 Opening and running the project

Opening and running the project requires these steps:

- You need to download and install the Android Studio. Accessible from `https://developer.android.com/studio/index.html`
- Download and install an SDK of version 14 and above and an Android Support Repository.
  - Click on the SDK Manager icon (see fig. C.1). The icon is located in the top bar of Android Studio.
  - Download and install one SDK from the selected options in figure C.1.
  - Download and install Android Support Repository as shown in the figure C.1.
- Open the project by clicking on File at the top left and choosing the Open option.
- The project can be run by using the Shift+F10 shortcut, or by clicking on the run icon.

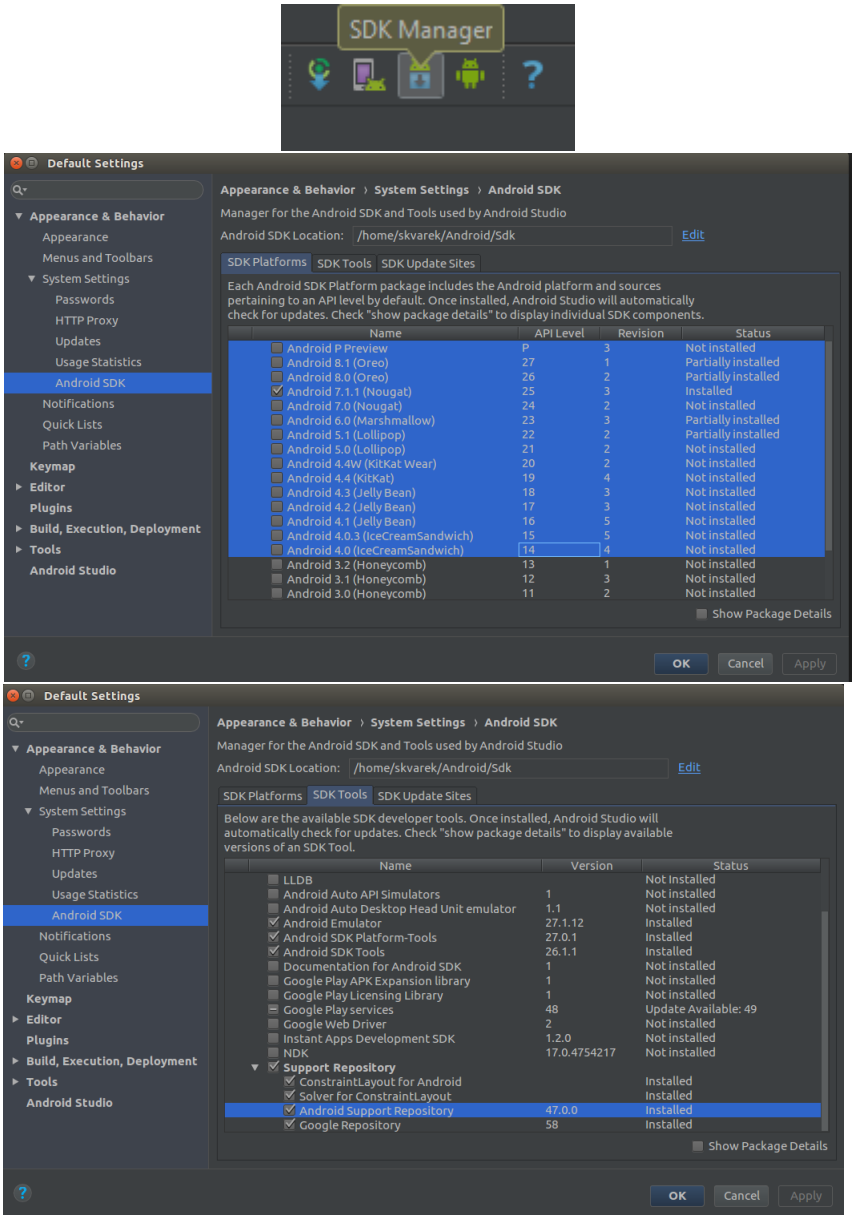

Figure C.1: SDK installation help

Appendix D

Contents of the enclosed CD

- `readme.txt` ..... brief description of the contents
- `apk` ..... a directory with runnable form of the application
- `src` ..... sources
  - `impl` ..... project with source codes
  - `thesis` ..... source code of this thesis
- `text` ..... text of this thesis
  - `thesis.pdf` ..... text of this thesis in PDF