Insert here your thesis' task.

Czech Technical University in Prague
Faculty of Information Technology
Department of Software Engineering

Bachelor's thesis

Android mobile client for an inquire system

Vojtěch Udržal

Supervisor: Dr.Ir. René van der Heijden

7th May 2015

Acknowledgements

I would like to thank my supervisor Dr.Ir. René van der Heijden for giving me the opportunity to develop this application and for the support during the development process and writing the thesis text. Furthermore, I would like to thank my university for enabling me to work on this project through the portal Cooperation with industry. Last but not least, I would like to thank my parents for overall support.

Declaration

Citation of this thesis

Udržal, Vojtěch. *Android mobile client for an inquire system.* Bachelor's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2015.

Abstrakt

Bakalářská práce se zabývá tvorbou Android aplikace určené pro vykonávání veřejných průzkumů. Hlavním cílem bylo vytvoření plně funkční a především spolehlivé aplikace, která bude fungovat i v offline režimu. Velký důraz byl kladen na jednoduché a intuitivní ovládání. Tato aplikace je vytvořena pro vznikající projekt Sprockler a byla vyvinuta s ohledem na jejich požadavky. Cíl práce se podařilo splnit a aplikace je využívána na prvotních projektech.

Klíčová slova Android aplikace, veřejné průzkumy, sběr dat, offline mód

Abstract

This thesis is dealing with a development of an Android application focused on public surveys. Main goal of this thesis is to develop a fully functional and reliable Android application, which can work off-line. Great importance is given to simple and intuitive control as well. This application is made for commencing project Sprockler and was developed according to its needs. The goal of the thesis was successfully completed and the application is already being used in some initial projects.

Keywords Android application, public surveys, data collection, offline mode

Contents

List of Figures

List of Tables

Introduction

In today's world, data of any kind play a key role in almost all fields. From strategic planning over monitoring and evaluation, we usually need data analysis to get useful answers. Yet, before we can do a data analysis, we must collect relevant and adequate data. Nowadays, in a modern and computer-driven society, data collection can be achieved with much more ease than before.

However, at certain situations, we cannot rely on or use data generated by machines and computers. For instance, if one wants to get some information about communities in third world countries, analysing computer data such as internet traffic will not make it. Much more valuable and accurate data can still be collected simply by interviewing people. By that we can collect their personal stories and opinions, which we would not be able to get just by analysing data that were produced by computers.

First of all, this thesis will discuss and evaluate platforms for running surveys with focus on their mobile applications.

Afterwards, the rest of the thesis will describe to the reader a development of a new Android interviewing application for a Netherlands-based project called Sprockler. Sprockler is at this moment a new emerging project found by Dr.Ir. René van der Heijden, who has a lot of experience in data mining and software development and Dr.Ir. Han Rakels MBA and MSc. Lisette Gast, who have great experience with running surveys in developing countries and communities. This application was developed in accordance with their needs and they will be referred to as stakeholders.

It is important to note that this application was developed through Co-operation with the industry portal and is licensed to Dr.Ir. René van der Heijden. From this reason, the source code of the application as well as its Doxygen documentation cannot be publicly attached to the thesis. However, it is available for inspection by relevant individuals (such as the opponent or committee members).

Chapter 1

Analysis

While developing a new software, it is always a good thing to look at the problem domain as a whole, analyse and evaluate all requirements, potential problems and, apart others, also a competition. In this chapter I will introduce the problem domain in detail and describe and evaluate similar applications. Moreover, I will specify requirements that were considered important for the new application.

1.1 Target user base

Since Sprockler is aiming at all situations where interviewing people makes sense, the target group of users of the Android application is truly boundless. The stakeholders have experience with running surveys especially in third world countries in Africa and South America, yet there are already chances that it will be used in Ireland or Amsterdam. From this I concluded, that a potential information literacy can vary from very experienced users to beginners in the fields of information technology. All these information had to be accounted for while designing the GUI, which will be described in later chapters. It is also apparent, that language abilities of users can differ dramatically as well.

With respect to the age of the users, it is expected to vary between 12 to 90 years old.

1.2 Similar Applications

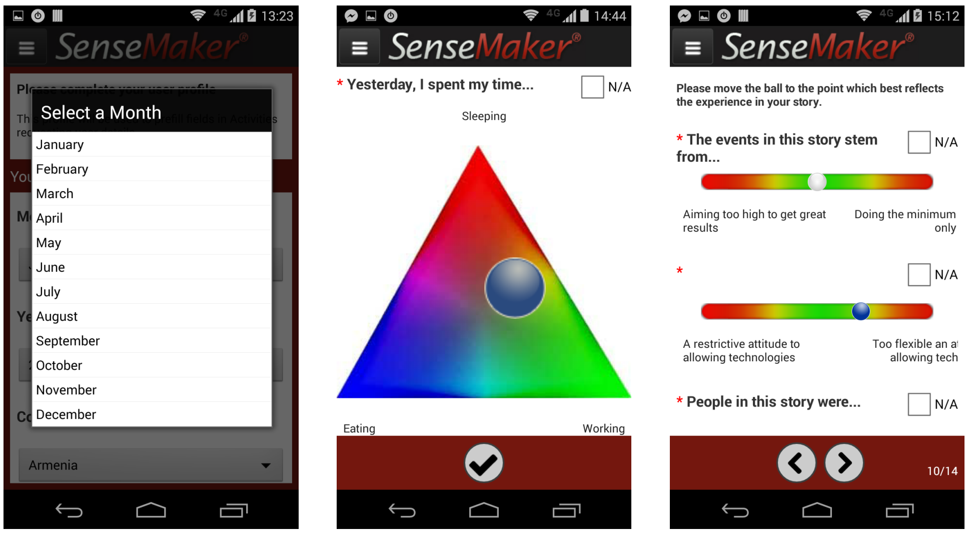
As already noted, analysing competition and their drawbacks is a key factor when one wants to develop a better tool. Prior to writing specifications for this application I examined several competitive survey platforms and focused on their mobile applications especially from a technological and graphical point of view.

1.2.1 Sensemaker

Sensemaker[1] was suggested by the stakeholders as a closest competitor and so I began my research with that. Sensemaker is part of an interesting project called CognitiveEdge[2], which was founded in 2005 with “the objective of building methods, tools and capability to utilise insights from Complex Adaptive Systems theory and other scientific disciplines in social systems”[2]. However, because I was focused on mobile applications I downloaded their *SenseMaker® Collector*[3] from Google’s Play Store and started testing.

1.2.1.1 Graphical user interface

Upon opening SenseMaker Collector I noticed that it did not look as a usual Android application. All of the GUI components looked different and unnatural, such as the months selection (see figure 1.1a). I also encountered several flaws in controls, for instance, the menu did not react on an edge swipe gesture and I had to click on the menu icon. This may seem as a minor thing but it definitely is unnatural for Android users.


13:23
SenseMaker
Select a Month
January
February
March
April
May
June
July
August
September
October
November
December
Armenia
14:44
SenseMaker
* Yesterday, I spent my time...
N/A
Sleeping
Eating
Working
15:12
SenseMaker
Please move the ball to the point which best reflects the experience in your story.
* The events in this story stem from...
N/A
Aiming too high to get great results
Doing the minimum only
*
N/A
A restrictive attitude to allowing technologies
Too flexible an a allowing tech
* People in this story were...
N/A
10/14

(a) Month selection (b) Triad question (c) Slider question

Figure 1.1: Unnatural Android GUI

Another point where Sensemaker goes against Android principles is the behaviour of back button. It is a common thing that Android application keeps a stack of previous pages and user can go back to them, however, when there are no more pages to go back to the application closes. Oddly, that is not how Sensemaker behaves, and instead it opens or closes the menu.

It is also impossible to overlook the poor-quality of some graphical elements, such as the triad triangle (see figure 1.1b). To my opinion the use of colors is not well considered and this applies for slider question as well (see figure 1.1c).

What I appreciated on Sensemaker was, that the screen layout always adjusted depending on the screen rotation. When in portrait mode, the buttons forward and backwards were at the bottom of the page as seen on figure 1.1c. Once the screen was rotated, the control buttons were moved into vertical bar on right side as illustrated on figure 1.2.

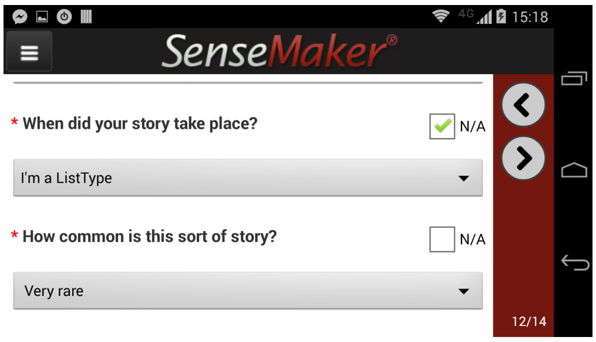

15:18
SenseMaker
* When did your story take place?
N/A
I'm a ListType
* How common is this sort of story?
N/A
Very rare
12/14

Figure 1.2: Buttons moved into vertical right bar

The sum of all, those imperfections results in quite an unattractive GUI. A great tool not only has to offer great functionality but must also be graphically attractive and intuitive to use. I took it as a challenge since I do not consider myself a person with a well developed graphical skills.

1.2.1.2 Technical functionality

While using Sensemaker, I had not encountered any application crashes or freezes. Nevertheless, I was quite sceptical about its reliability. From time to time the application just stopped, pretending to be doing some task for several minutes without ever finishing them. One of those cases can be seen on figure 1.3, where the application stayed like that for several minutes and the task had to be canceled.

1.2.1.3 Summary

Considering the above, I came to the conclusion that Sensemaker is not a native Android application, but uses Cordova [4] framework instead. This conclusion was further substantiated by the fact that, at least in one occasion, some partial HTML tag remained visible in the GUI. More details about this technology will be mentioned in Selecting Application's technology chapter 2.1.1, but I can already state that having a look at Sensemaker was a key reason why I have decided not to use it. Furthermore, the mentioned glitches certainly did not make a great impression and I knew it was important to prevent any minor defects in our application.

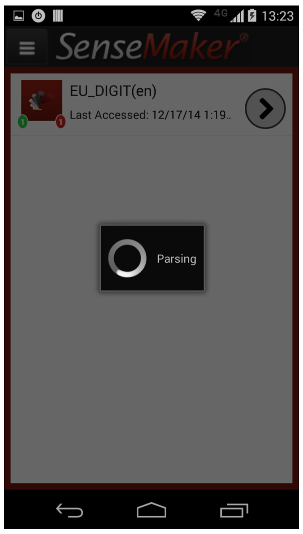


Figure 1.3: Infinite loop

1.2.2 SurveyGizmo

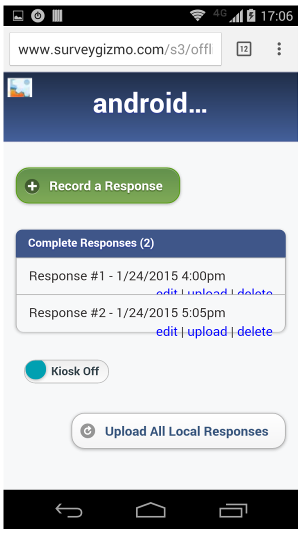
Another popular and large survey platform is SurveyGizmo. It was founded in 2006 and is based in Colorado, USA. As one can read on the website, their mission is to "provide great survey software and great service to businesses and organizations around the world." While they offer wide range of different question types, they do not support special questions such as Clickable images or Tripoles, which were important for the stakeholders and will be introduced later.

1.2.2.1 Graphical user interface

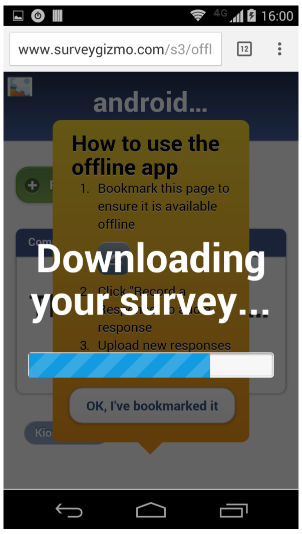
The graphical design of SurveyGizmo's mobile web application is clean and intuitive, even though we can still encounter minor imperfections such as the edit, upload and delete buttons overlay on figure 1.4a. All in all, from graphical point of view, SurveyGizmo comes out much better than SenseMaker 1.2.1.

1.2.2.2 Technical functionality

SurveyGizmo supports offline mobile surveys, but only to a limited extend. Instead of native Android or iOS application, which would be available for download in Play Store or iTunes, they offer the possibility to download and store web page and survey's data in a browser's cache (see figure 1.4b). That brings a great advantage of reusability of such a solution, since the same code will work well on all devices with browser regardless the platform, and thus makes the development much faster and cheaper.


www.surveygizmo.com/s3/offl
android...
Record a Response
Complete Responses (2)
Response #1 - 1/24/2015 4:00pm
Response #2 - 1/24/2015 5:05pm
edit | upload | delete
Kiosk Off
Upload All Local Responses

(a) Minor imperfections: button overlay


www.surveygizmo.com/s3/offl
android...
How to use the offline app
1. Bookmark this page to ensure it is available offline
Downloading your survey...
3. Upload new responses
OK, I've bookmarked it

(b) Caching of survey in browser

Figure 1.4: SurveyGizmo web based application

1.2.2.3 Summary

Even though I like the simplicity, elegance and quality of the SurveyGizmo's mobile solution, it is not suitable for our application. There are several drawbacks when it comes to browser based applications, one of them being the lack of access to special services of the Android platform such as Alarm Manager. Alarm Manager is used for setting timeouts of a long time repetitive tasks such as upload of responses. Since it is not accessible from browser based application, the responses must be uploaded manually by tapping a specific button. That makes it inconvenient and puts unnecessary task on the user.

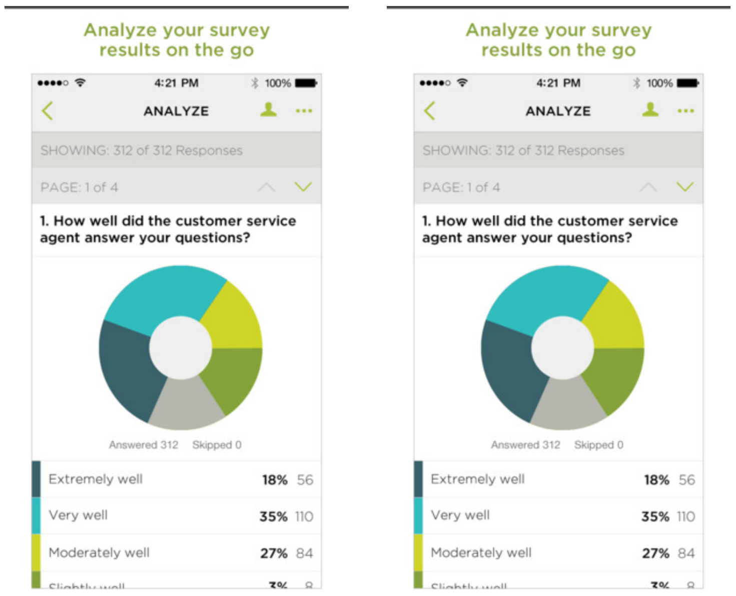
Another disadvantage is, that there is no list of downloaded surveys. It is responsibility of the user to bookmark a specific survey page so he can load it when offline, which is also unnecessary and confusing. Both mentioned disadvantages are really just a beginning of all the limitations that browser cached applications have.

1.2.3 SurveyMonkey

Last platform I focused on was SurveyMonkey. It is a huge company with headquarters in Palo Alto, California and over 10 years of experience. They are proud to state that their "customers include 100% of the Fortune 100, as well as other businesses, academic institutions, and organizations of all shapes and sizes." and that "millions of people use SurveyMonkey " [5].

Given the size of a company, I was expecting them to cover whole market from iOS over Android to Windows mobile platform. I was surprised that even after thorough examination of the web site, I only found a reference to an iOS application [6]. There is no sign of Android, even though there is a demand for that, as their website says: "We're listening–and we know you want a SurveyMonkey Android app..."[6]. As a solution, they claim that their surveys use responsive web design so they will look well on any platform. However, their web surveys are not as advanced as SurveyGizmos's, so they do not work offline.

On the other hand the iPhone and iPad application look amazing 1.5, but unfortunately I did not have an opportunity to test it.


Analyze your survey results on the go
4:21 PM
100%
ANALYZE
SHOWING: 312 of 312 Responses
PAGE: 1 of 4
Answered 312 Skipped 0

1. How well did the customer service agent answer your questions?

Extremely well	18%	56
Very well	35%	110
Moderately well	27%	84
Slightly well	3%	8

(a) (b)

Figure 1.5: SurveyMonkey iPhone application

1.3 Functional Requirements

I briefly discussed the problem domain over a Skype interview with the stakeholders, but specific requirements were agreed on on a first meeting in The Netherlands. From that, and from analysis of similar applications we have come up with the following list of requirements for the application:

F1 Download surveys User will be able to download a survey upon entering survey code.

F2 Update surveys The application will provide a possibility to update surveys.

F3 Offline mode The application will be able to work offline and provide full functionality for downloaded surveys. Once connected to the internet, collected responses will be uploaded to the server and removed from the application.

F4 Multilingual The application will be able to present itself in different languages, among which are English and Arabic. User will be able to switch between languages that are supported by the downloaded surveys or to a language of the Android system (if such a language is supported by the application).

F5 Conditional logic The application will support simple conditional logic. For example, skipping of certain questions based on responses to previous question depending on how the rules specified for the respective survey.

F6 Saving of common questions's responses The application will save responses to questions that have a specific tag into device's memory. When applicable, such stored responses will be used as default answers to any subsequent question with the same tag. Depending on a design option, it will either show the question so the user can confirm the answer or completely skip the question and answer it automatically. The user will be able to see all saved responses and question tags that are stored on the device and must be able to individually delete them. This tagging system forms a flexible alternative to user profiles that other systems may use.

F7 Questions help At complex question types, the application will offer a help screen explaining that type of question.

F8 Question illustrations The application will be able to display question's image depending on how it was designed.

F9 Required questions The application will not let the user continue or complete the survey unless required questions are answered.

F10 Questions order and grouping The application will support grouping of questions on one page and display questions in the order they were designed.

F11 Interviewer mode The application will have protected interviewer mode, which will allow the application's user to collect responses of other people.

F12 Question types The application will support following question types

1. **Open question** Depending on how designed, this question will offer the possibility to write a text responses, take and select a photo and video and record audio.
2. **Multiple choice question** Depending on how designed, it will allow selecting specified amount of options or may also offer a possibility to write the response in a text field.
3. **Bipole** It will show a slider which user can drag and set between two options, depending to which he gives more value.
4. **Tripole** Triangle with 3 options at each corner will be shown. User will be asked to move a dot inside the triangle to a position where according to him it strikes the best balance.
5. **Clickable image** An image will be presented and user will be asked to point one or more locations.
6. **Descriptive question** This question will only be used as an informative text for the user and will not offer the possibility to respond.

F13 Mobile data limit The application will make it possible to set a data limit on uploaded responses so user's mobile data plan is not drained by uploading large media files. The limit will not be applied when connected over WiFi.

F14 Not applicable option The application will provide *Not applicable* option for questions which allow it.

F15 Delete survey User will be able to delete downloaded surveys.

F16 Storage of location and time The application will store a location where the inquiry has been filled and a time, when it was started.

F17 Randomization of choices The designer can instruct the application to randomize the presentation order of the options, which may help to avoid bias due to a specific order. The application should respond to such an instruction.

F18 Interviewer questions The application will mark questions which are targeted at the interviewer. Such a question can be used to record the interviewer's impression of the interviewed person.

1.4 Non-functional Requirements

NF1 Running on Android OS The application will be running on Android powered mobile devices.

NF2 Simple, easy and intuitive design Given the target user base, the users should be able to grasp control with ease from the very beginning. In order to achieve that, Android Design guidelines [7] should be followed. [1]

NF3 Starting application via a link The application will be started and user will be taken to survey download page when a link with a survey code targeting Sprockler's website is clicked.

NF4 Encryption of responses The application will encrypt responses and safely transport them to the server.

[1]Application was implemented in the Summer of 2014, so it does not follow later released *Material design*

Chapter 2

Technical Design

Technical design is the second main step in software development, after specifying the functional and non-functional requirements of the application. The challenge here is to translate these requirements (the what part) into a suitable design (the how part). Careful design of application can help to avoid difficulties at the implementation stage or in future development.

2.1 Selecting technology

Selecting the right technologies is one of the key factors that shape the software for its whole life. A wrong decision can have devastating consequences. Every software developer usually stands in front of this decision multiple times in a software project, and this case was no different. I had to select the right technologies for the application, for the communication protocol, and for the data storage.

2.1.1 Application

As one can read in *Similar Applications section*1.2, there are different ways of how to create an Android application and each has its own up and down sides. The first decision concerns the required Android version. Since less than 8% of Android devices run a system version lower than 4.0 (API 15) [8], it was decided to overcome possible compatibility issues by only supporting system versions above API 15 (Ice Cream Sandwich and newer). Apart from other extensions, version 4.0 has better multi lingual support which is key for the application under consideration. In contrast of what is generally presumed, smart phones in developing countries are quite modern, making this limitation acceptable to the stakeholders.

2.1.1.1 Cordova

One way to make an Android application, is to use the Cordova framework [4], that allows developers to create a mobile application using web technologies such as HTML and Javascript and provides them a limited API to access some of the native device's functions. Cordova, also known as Phonegap, was founded by Nitobi Software and acquired by Adobe in 2011 [9]. If a developer wants to use more advanced features of the native Android API, he must create a custom Javascript plugin which provides a bridge to the Android API. The Cordova application can be understood as single page application (SPA) wrapped by a device platform's web view. Using this framework offers the undeniable benefit of cross compatibility between different platforms. The developer can, with a few changes, deploy the same application to Android, iOS, Windows Phone, and other platforms. Furthermore, he does not need to learn each platform's different programming language such as Java for Android and Objective-C for iOS. On the other hand, he should at least know about one of Javascript SPA frameworks (such as *AngularJS*, *Backbone* or others) and web design.

In the beginning, we have also been playing with the idea to develop the application in the Cordova framework. The idea to *develop once - run everywhere* sounds just too nice to not try it. Moreover, it was specifically suggested by the stakeholders. However, after evaluating performance, reliability, and graphical interface of applications running on Cordova, it was decided not to use it. At times, they performed slow, sometimes behaved in an erratic way (see figure 1.3) and did not comply to the Android design guidelines as mentioned in section 1.2.1.1. Thinking about this, the whole idea that is summarized in *develop once - run everywhere* seems impossible, as iOS and Android guidelines simply do not match. Last but not least, even though the community behind Cordova is very active, more than a few of the plugins and extensions I tried for accessing Android's API, which are necessary for any non-trivial tasks, were bugged or did not work well and I did not want to get clogged by fixing other developers's bugs while developing this application.

2.1.1.2 Xamarin

Xamarin enables developers to write a complete application using C# and use up to 90% of the code on all platforms [10]. It is a very interesting project that looks like a solution to diverse mobile platforms development and seems like a big relief for developers. It was founded in 2011 by creators of Mono [11] in order to unify the development of mobile applications. Unlike Cordova, which makes your application look exactly the same and unnatural on all platforms, Xamarin let's you share only your business logic, models, data layers and similar. All user interface implementation is still developed in C# , but must be implemented individually for each platform. Therefore, the programmer needs to know his way on both platforms, but can still save a lot of time by reusing significant parts of the code. As a result, the application looks natural and native on all platforms, while the costs of development can be kept lower, depending on the project. However, Xamarin comes at a price which was one of the reasons we did not choose it. It costs about $25 per month per developer and per platform [12]. Furthermore, the community around Xamarin is not big enough yet, which may form an obstacle when looking for help with a problem or bug. The small community makes such an endeavor much less fruitful.

2.1.1.3 Android SDK

Even though there are many ways how to develop an Android application, using Android's SDK is still the most popular one. It has several benefits which outweigh the disadvantages for many developers including ourself. First of all, it is the oldest way of developing Android applications, so it contains fewer bugs and has a solid documentation with many best practises and tutorials [13]. Moreover, there is a massive community of tens of thousands developers who publish tutorials or answer questions about Android's SDK. This gives the developer a strong confidence that he will not get stuck with some problem or bug which would take days to fix. Last but not least, the SDK provides full functionality of the platform and is recommended by Android. This altogether supported our choice for the Android SDK.

2.1.2 Data storage and communication

I have decided to store surveys and responses on the device by using SQLite databases, which is a common practice for the Android platform, while the Android's SDK provides a convenient library for working with these databases [14]. The SQLite database containing survey will be created by the server and downloaded by the application as regular file upon HTTP request. As such, SQLite file will be also used as a container for transportation. A similar approach will be used for collecting and uploading sets of responses (referred to as participations), where every participation will use its own SQLite database that is uploaded via regular HTTP post request. The JSON format will be used for other communication between the application and server, such as a check for updates.

2.2 Modes of operations

One of the requirements (F11) was to provide two different modes of operation: an interviewer mode that will allow to collect an unlimited number of participations from general participants, and a private mode that will allow to collect a single set of responses (typically from the owner of the device). It was decided to meet this requirement by using different survey codes and make the distinction by their code length. Therefore, the interviewer mode can be accessed only by people who have an interviewer survey code for the particular survey. Apart from the interviewer mode and the private mode, we decided to add a test mode that can not only be used to test the application, but also to test-run new surveys. Each survey mode has slightly different purposes and allows different actions. Here I would like to briefly describe the survey modes and show their supported actions on figure 2.1.

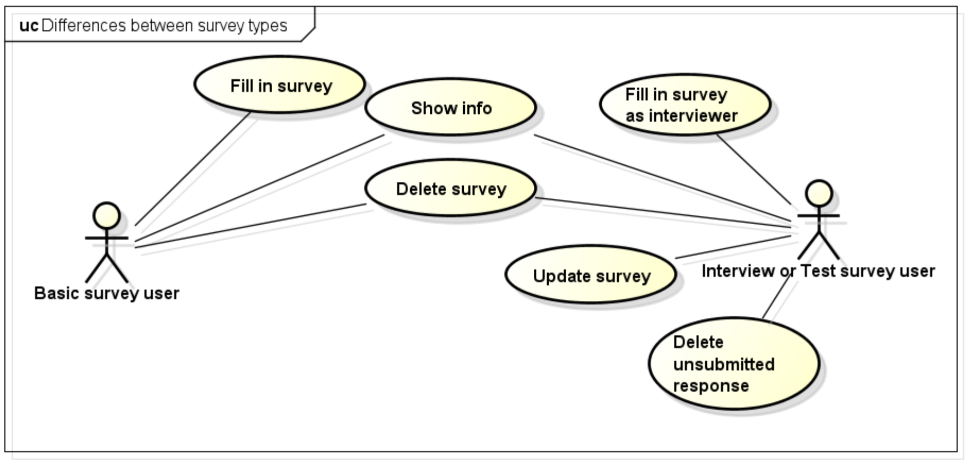

uc Differences between survey types
Fill in survey
Show info
Fill in survey as interviewer
Delete survey
Update survey
Delete unsubmitted response
Basic survey user
Interview or Test survey user

Figure 2.1: Survey type actions

Personal mode Personal mode is expected to be the most common survey type. Its survey codes consist of 5 characters and can only be filled twice. This survey type is intended to be used by general public when a survey is launched and the survey code will usually be distributed.

Interviewer mode Interviewer mode is supposed to be accessed and downloaded by selected people only. Its survey codes consist of 6 characters and will offer the possibility to run the survey for an unlimited number of times. This survey mode will be used for collecting responses from different people, e.g. by having an interviewer going from door to door. In case the same person participates multiple times, the participations will share a unique code so all the responses can be connected and allow for an appropriate analysis.

Testing mode The purpose of testing mode is to provide a possibility for survey designers to try the survey on a mobile device without influencing the results. While testing, the responses will be uploaded, but the server will mark them with a special tag.

2.3 Use cases and their scenarios

Use cases describe the functionality of a software from a user perspective. Each Use Case describes a specific interaction between a user and the application. It works best for functional requirements which are rich on user interaction, but it does not do a good job in expressing non-interaction functionalities. Therefore, following use cases only list interactive functions, and disregard other functionalities.

UC1 Download survey User can enter survey's code and download the survey.

1. The user will click on a button on main screen to add a new survey.
2. The application will show a page with text field.
3. The user will enter survey's code to the text field and click *download*.
4. The application will show a progress dialog indicating the status of the download.
5. If the download was successful, user is taken to the main screen. If the download failed, an error message is shown.

UC2 Update survey User can update surveys.

1. The user will long press on a survey.
2. The application will show context menu for that survey.
3. The user will select *Update* option and application will take him to download page with pre-filled survey's code.
4. Following steps are similar to UC1.

UC3 Upload responses Even though responses are uploaded automatically, the user is also able to invoke the upload process manually.

1. The user will press and hold a survey.
2. The application will show a context menu for that survey.
3. The user will select the *Upload* option.
4. The application will show an indeterminate progress bar and then update completed/uploaded statistics.

UC4 Show questions help Complex questions provide a special help dialog to describe the question and how to answer it.

1. The user will open the complex question and a question mark will appear in top right corner.

2. Upon tapping the question mark, a dialog with instructions will be shown.
3. The user will close the dialog by tapping *OK* button.

UC5 Possibility to disable or restrict uploaded files over mobile data The user is able to disable or restrict mobile data usage for uploaded files.

1. The user will open the menu drawer and select *Settings*.
2. The application will show the settings page.
3. The user will either check or uncheck *Use Mobile data* or *Small files only*, depending on his preference.

UC6 Not applicable responses Some questions can be answered by checking Not applicable checkbox.

1. A question will be shown with the Not applicable checkbox in top right corner.
2. The user will check the checkbox and continue to the next question.

UC7 Change language User can change the language of questions and the application.

1. The user will click on the Globe icon in top right corner in action bar.
2. The application will show a list of available languages that at least one of the downloaded surveys supports.
3. The user will select a language by tapping on it and navigate back by pressing back button.
4. The application will return to previous page.

UC8 Show saved responses User can look at responses and their tags that are saved on the device.

1. The user will open the navigation drawer and select *User Profile*.
2. The application will show a list with *Tag* and *Answer* column.

UC9 Delete selected responses User can delete saved responses on device.

1. Include (UC8 Show saved responses)
2. The user will check the answers he wants to delete and click *Delete selected answers*.

	F1	F2	F4	F6	F7	F13	F14	F15
UC1	X							
UC2		X						
UC3								
UC4					X			
UC5						X		
UC6							X	
UC7			X					
UC8				X				
UC9				X				
UC10								X

Table 2.1: Functionality coverage by use cases

3. The application will remove those answers from the device.

UC10 Delete survey User can delete downloaded survey.

1. The user will long press on a survey.
2. The application will show context menu for that survey.
3. The user will select *Delete* option.
4. The application will show a confirmation dialog.
5. The survey will be delete upon confirming the deletion.

2.4 Function requirements covered by use cases

Most of interactive functional requirements are covered by use cases. It can be seen in table 2.1

2.5 Software architecture

Every software program should have some software architecture which is efficient, maintainable, readable and functional. Even though it may seem time consuming in the beginning, it is always worth the time to put some thought into it, especially when the application is supposed to be complex or continually developed, and/or to be supported by other people. The Android platform does not demonstrate any preference or enforce any software architecture for its applications. Accordingly, it is up to the responsibility of the programmer. I decided that the base architecture should be the model-view-controller (MVC). The MVC pattern separates the business logic from the user interface and is well known throughout the developers community. Readers who would like to learn more are referred to e.g. [15] or any of the other online sources.

Since there is no direct support of Android SDK for any software architecture including MVC, programmers need to match Android's provided classes and elements to the selected architecture pattern. Fragments, Activities and Adapters play the same role as Controllers. XML view layouts and View classes are the same as Views. Finally, Models are created by the programmer as any class or entity that encompasses main business logic. The final architecture in the application may not always follow the true principles of MVC because it had to be adapted to the Android platform and should work with the classes provided. For instance, some models are bound directly as *Listeners* on their views in order not to clutter Fragment class code, even though that does not precisely respect the MVC pattern.

2.6 Questions model

Since the application will have to support many question types, it is necessary to have a reasonable model design. I have decided to create an abstract class Question, from which the other questions are inherited (see figure D.1). Furthermore, after looking closer on the question types, I have created another abstract class OptionQuestion, which is a parent of any question type which has some options (i.e. RadioQuestion, SliderQuestion, Tripole, Clickable Image, CheckboxQuestion and ComboboxQuestion). Creating of question instances is handled by QuestionFactory, which has a method create() and accepts a question type, question id, survey database and participation database as parameters.

An abstract Question class implements interface Printable, which is used by Visitor pattern [16] for displaying the question. Additionally, QuestionWithTag interface is implemented by question types which allow storing of responses into device's permanent memory.

To enable multiple questions on a page, questions are grouped in a *QuestionGroup* class. Each question group represents a single page of the survey and is created by *Survey* model. An interaction between *Controllers* (SurveyFragment and QuestionListAdapter) and *Models* (Survey, QuestionGroup and Question) is outlined on figure D.3.

2.7 Database model

Even though the server side is not part of this thesis, it was my task to design a database structure for storing data on device and for transmitting them from or to the server. There are three different database structures which will be described in following subsections.

2.7.1 Surveys Manager database

We would like to recall the fact that each survey has its own SQLite file and there is no central database on the device which would store all information about downloaded surveys. This approach was used for performance and concurrent access reasons. The only exception is the SurveysManager database. It is a small database which has following three tables and stores only basic information about the downloaded surveys.

- **Surveys** This table stores some general information about all downloaded surveys. For instance, the survey code, version, completed and uploaded statistics.
- **Languages** Stores languages which are supported by downloaded surveys. It is updated when a survey is downloaded or removed and is used for showing available languages on language change page. It lists all supported languages per downloaded survey.
- **PersistentQuestions** Stores responses on questions which have a tag as described in functional requirement F6.

2.7.2 Survey database

Survey database is used for storing all information about a survey, such as questions, translations, images, exclusion logic rules, and so on. There is one database file for each survey, and it is created on the server and downloaded by the application. The structure of the database is as follows:

- **Languages** Stores languages which are supported by this survey. Those rows are added to Survey Manager database upon download.
- **Questionnaires** Stores information about the survey. It always contains just one row.
- **QuestionnaireTitles** Stores translations of survey's title.
- **Questions** Stores survey's questions.
- **QuestionTexts** Stores different translations for questions.
- **QuestionOptions** Stores options for option questions and their translations.
- **QuestionExclusionLogic** Stores rules used for hiding or showing questions.
- **QuestionIllustrations** Stores question images in a blob field.

2.7.3 Participation (Responses) database

Each participation has its own database and thus, a separate SQLite database file. This is to ease concurrent access especially in situations such as uploading of previously saved and completed participations while user is saving responses to other participation. By splitting each participation to separate database(file), the submit process can treat the responses like files and not like rows in one database. Upon successful upload, the particular responses database is deleted. Structure of the database is following:

- **Participation** Stores general information about the participation such as location and start time.
- **Responses** Stores responses to all questions except option questions. It can store text and numbers.
- **ResponseOptionsSelected** Stores which options were answered. For Tripoles, it stores also distance to the option and for Clickable image a coordinates of the option.
- **ResponseMedia** Only audio recordings are actually stored in the database. Videos and photos are stored as files because of performance issues and this table is used only as pairing table between the responses and the files on disk. It turned out that when SQLite has more than 30MB, it gets significantly slower. Furthermore, the data itself can be then uploaded over a mobile network in an early stage, while the media will be added later when a WiFi connection is available.

2.8 Communication with server

As already mentioned, the application needs to communicate with a server to download surveys, and to upload responses. However, the application will not be dependent on internet connection, so user can flawlessly work with already downloaded surveys off-line. Responses will be stored in separate directories, called participation directories, which will always contain one participation SQLite database (see 2.7.3) and then photos or videos, which the user supplied. Photos and videos, which will generally be over a certain size (approx. 4MB), will be split into smaller file parts in order to make the upload smoother in case of an unreliable internet connection. It also eliminates the need to be restarted every time the connection has been lost.

2.8.1 API description

In order for the application to communicate with a server, the server must support following requests.

GET /downloadSurvey/{survey code}/{application version}/{downloaded survey version or 0} This request is used for downloading or updating a survey. This request has following response codes:

- **200 OK** When code is valid and survey is found. The survey SQLite file is sent in the response body.
- **304 NOT-MODIFIED** When the downloaded survey is the newest version already.
- **404 NOT-FOUND** When survey with this code does not exists or is not available.
- **460 NOT-COMPATIBLE** When this application's version is too old and not compatible with the survey database file. The application notifies the user to update the application.

POST /participationSubmission/{code} This request is for uploading of responses. Each file (such as participation database, media file or its filepart) has its own participationSubmission request. Server responds either with 200 for accepted or 500 for error. In that case, the file is kept on the device and the application will attempt to upload it later.

POST /checkForUpdate This request is for checking available versions of downloaded interviewer surveys, because surveys can be changed and published with a new question after its older version was already downloaded.

Request The request must contain a JSON array with JSON objects, one for each downloaded survey. Each object then contains survey code (*code*) and downloaded version of the particular survey (*version*) as well as a version of the application itself (*appVersion*).

Response The response is a JSON array of objects each containing a newest version number (*latestVersion*) and a survey code (*code*). If a new version is available, the application shows an update icon next to the survey.

2.9 Security

Security was important for the stakeholders (NF4), because the application is going to be used in various types of environments and societies where not all of them are safe. It was necessary that the data are encrypted from the moment participant fills out the survey and that neither interviewer nor anyone else who gets control over the device will have ability to view previous responses. Because this thesis text is public, it is in stakeholder's interest that some specific security details are omitted.

2.9.1 Communication security

Communication will be realized over HTTPS protocol with a certificate authorized by a well known certification authority trusted by Android devices. When using HTTPS with authorized certificate on Android, the implementation's code does not differ from usual HTTP communication except for the used protocol name in address.

2.9.2 Device data encryption

Encrypting communication is important to avoid eavesdroppers on the network, however, it will not help when someone gets access to the device. For that, we will need to encrypt the responses on the device already. Android offers a way how to achieve that easily by simply turning on encryption for the whole device. We discussed this method with the stakeholder as most reliable one, because the whole device is encrypted and no-one can take control of it unless they know the password. Nonetheless, after this consultation it was decided that a mechanism to encrypt responses data should be implemented independent of the device settings.

One of the ways to achieve that would be by using synchronous block cipher for file encryption with a key, which would be asynchronously encrypted by a public key and transported in the beginning of the files. Synchronous encryption should be done probably by AES with 256-bit key, which is a recommended standard.

Chapter 3

User interface

For the design of the user interface, I used a wireframe designer FluidUI [17]. It is a neat tool that helps you to quickly and easily create wireframes for your application. It already contains predefined Android elements and templates, helping to make the wireframes look very close to real Android application. I have been designing the user interface for the initial meetings with the stakeholders in order to elucidate and communicate my understanding of the application. Reversely, it allowed the stakeholders to get an early but quite realistic glimpse of what the application could look like. Equally, this enabled an immediate adaptation, even before starting with the technical (architecture) design. Last but not least, it also turned out to be very convenient to print the wireframes in a size common to mobile devices, and to take a look at them in hand. I consider this as a great benefit and recommend others to use the same approach as it facilitates discussing the application with the stakeholders and clarifies what everybody has in mind by making the concepts and ideas much more concrete. The main priority was given on mobile design, however, from time to time I also adjusted the design to better accommodate tablet devices.

3.1 Action Bar

The Action bar component was introduced in Android 3.0 (version 11) but thanks to Android Support Library [18] it is available on all Android versions down to version 7. The action bar component has become an essential part of most applications, and provides similar functionality across all of them. It allows a user to better grasp control of an application that he encounters for the first time. The action bar component is always located on top of the screen, and starting from left usually contains an icon of the application together with an icon to access the hierarchically higher pages. Then, there could be a textual representation of a visible page or section, so the user can always see in which part of the application he currently is. Then, there are usually a few but most important icons for specific actions which are called "Action buttons". Lastly, there can be a three dot overflow icon on the right, indicating that additional actions are available from the dropdown menu, that the overflow icon shows or hides. The action bar is configurable allowing a developer to choose to display only those elements that he finds relevant or convenient for the application.

The stakeholders required that changing a language would be easy to do regardless of application's page. To accomplish that, language selection was an obvious choice for the Action button in the Action bar (see figure 3.1). Furthermore, we added a drawer navigation icon to the left, which indicates that there is a drawer to be opened from the left side. The drawer navigation itself will be described in following section. In addition, there is a text that describes the currently opened page, such as Download for download page or the survey title when in a survey. Only in the case where detail views of spe-

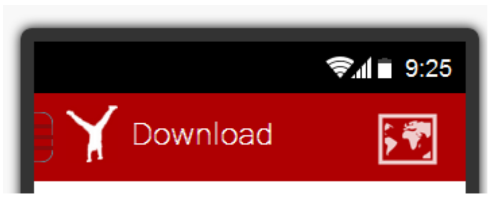


Figure 3.1: Wireframe of actionbar

cial question types (like Tripole or Clickable image) require so, the contents of the Action bar changes. Those questions are always opened in a separate page and it was necessary to create a different action bar, that provides buttons for saving the answer, canceling the answer and showing a help dialog. An example of such an action bar is presented in figure 3.2.

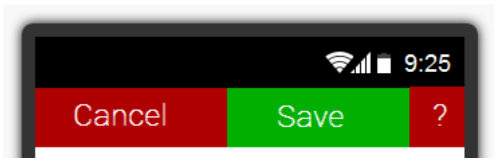


Figure 3.2: Wireframe of question detail actionbar

3.2 Navigation drawer

The navigation drawer is a panel that is shown after a swipe gesture from the left edge, or by clicking drawer icon in action bar. The navigation drawer should provide a quick navigation to different screens of Android application. Together with action bar's action buttons, it basically forms a main menu. The first item of navigation drawer is a link to main screen, which contains a list of all downloaded surveys. Next is the *User profile* section, that shows information about stored answers. *Language*, *Settings* and *About* items follow next. The wireframe of the navigation drawer is shown in figure 3.3.

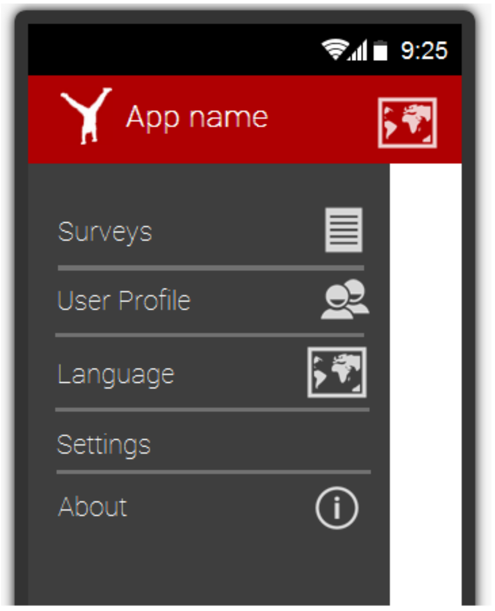


Figure 3.3: Wireframe of navigation drawer

3.3 Download survey screen

To start using the application, a survey must be downloaded first. When the user navigates to the Download screen from the surveys list, he will be presented with a text box to enter a survey's code and a Download button. Naturally, after entering the code and pressing the button, a progress dialog will be shown indicating the download progress, and reporting any errors. If everything went smoothly and the survey was downloaded, the user will be taken back to surveys list screen.

3.4 Surveys list screen

The home screen of the application will be a list of all downloaded surveys, providing quick navigation to the application's main functionality - filling out inquiries. When there are no downloaded surveys (such as when the application is ran for the first time), an informative text is shown with instructions how to download a survey: by pressing a plus icon at the bottom of the survey list screen, which links to the download page. The list itself displays some information about surveys, most prominently, surveys' title and subtitle. Because the application supports three modes of operation (personal, interviewer and testing), the survey mode is displayed as well. It is indicated by means of a specific icon for interview mode (microphone), and for test mode (repair icon). Next, there are two numbers showing how many surveys have already been completed and how many have been uploaded so far. This is especially helpful for interviewers. A progress bar reflects the completed and uploaded numbers, but also works as a divider between the listed surveys (see figure 3.4). Last but not least, an edit pen icon indicates which surveys have been started but not yet completed.

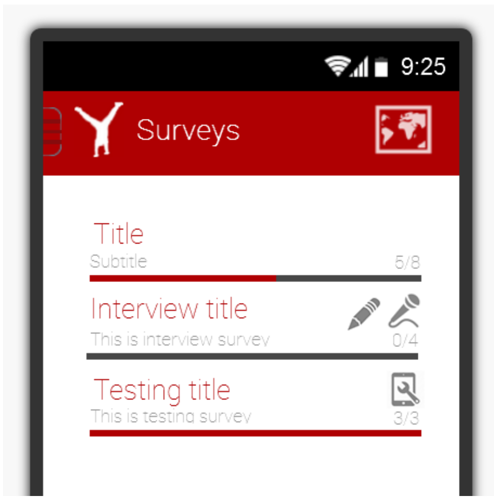


Figure 3.4: Wireframe of surveys list

It is a common practice on the Android platform to show additional actions for each list item. Even though it is not visible in any way, users familiar with the Android platform expect a context menu for each list item, that can be invoked by a press and hold action on the respective list item. Of course, I will comply to this standard and display a context menu after the press and hold action on any survey. From that context menu, a user will be able to:

- Restart the survey from the beginning.
- Immediately submit responses (and not wait for the automatic upload).
- Delete a survey.
- Delete a completed but not yet uploaded set of responses.
- Reset the build-in completed and uploaded counters.
- Update a survey - this takes user to download page with a pre-filled survey code.

- Display information about the survey (code, version and available version).

3.5 Survey screen

When a user taps on a survey from the surveys list, the respective survey will start, and he will be able to answer questions. Questions will be grouped on pages and ordered in a list, conforming the design of the survey. The exact behaviour depends on the design of the survey. Each screen has two control buttons at the bottom, allowing to navigate forward and backward. The bottom of each page also shows a progress bar that indicates, how many questions are still left. The forward button will be disabled if some yet unanswered question requires an answer. Users can move forward without answering the question when the designer allows to skip that particular question. In case the designer allows the user to indicate that a particular question is not applicable (so called Not Applicable or N/A flag), the application shows a checkbox at the top right of the question. If user checks this N/A box, the layout of such a question will be hidden.

There are several complex question types, which required a careful design to be sure that they are functional, simple and easy to understand. Usually those questions cannot be answered from the questions list but require an additional tap to take the user to the individual question screen instead.

Selected question types will be described in the following subsections.

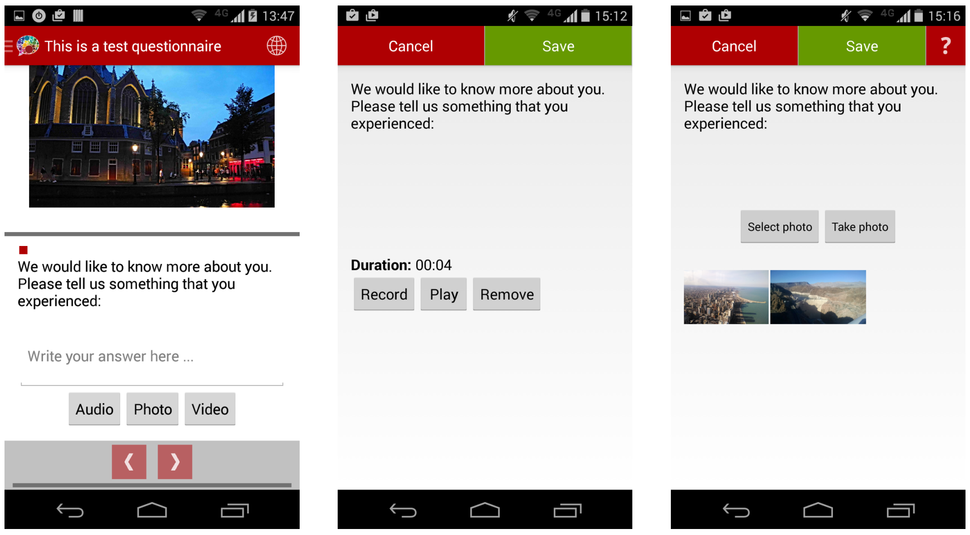
3.5.1 Open question

Depending on survey's design options, an open question can be answered by a text, an audio recording, a photo, or even a video recording. It would be to confusing and messy if all those functionalities were shown at the questions list. Therefore, open questions only offers to type in a text response from the questions list page, while responses by audio, photo or video are offered by their respective buttons that link to specific screens (see figure 3.5a).

The audio recording screen shows the question text on top and displays buttons to record, play, stop and remove a recording (see figure 3.5b).

The photo detail screen also shows the question text on top, while the user is provided with buttons to select a photo or take a new photo. The button to select a photo starts the gallery application, where user can choose the photo he prefers, which is then added to the screen. Naturally, taking a photo starts the camera and attaches the photo as well. A photo can be removed by tapping and holding it to open the context menu, and then selecting *Delete*. For this question type, the user is able to click on a question mark at the right of the action bar. This will invoke a help dialog with some explanation of the question (see figure 3.5c).

The video detail page behaves exactly the same as a photo detail page.



We would like to know more about you. Please tell us something that you experienced:

(a) Main layout (b) Audio detail (c) Photo detail

Figure 3.5: Open question screens

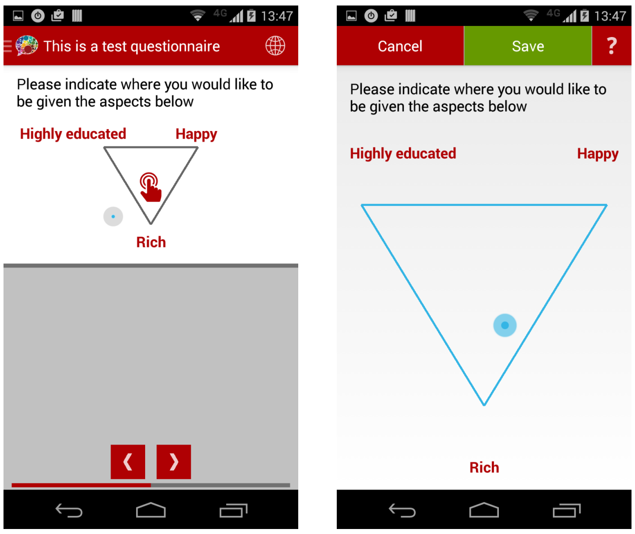
3.5.2 Tripole question

The Tripole question is one of the most interesting question types. It shows a triangle and a touch point in a question layout. The actual answering process requires a detail screen because that leaves more space for the user to select an appropriate point in the triangle. If there is no answer or if the answer is invalid (out of triangle), a ”touch me” icon will be shown in the questions list (see figure 3.6a).

When the user taps on the Tripole in the questions list, a detail screen is opened. This detail screen allows the user to drag the circle and place it to a position where it optimally reflects his opinion. Every time the circle is dragged over the edge of the triangle, the device vibrates allowing the user to place the point on one of edges or in the corner of the triangle. Also, the triangle sides will turn from blue to grey when the point gets out of the triangle and vice versa.

3.5.3 Clickable image

The Clickable image is an innovative and also very interesting question type which needed a good thought in terms of design. Similar to the Tripole question, the Clickabe image question shows a ”touch me” icon in the questions list when it is not answered (see figure 3.7a). Upon clicking the image in the questions list, a detail screen will be launched which presents the image and the question text, as well as a list of options (max. three) that need to be placed in the image (see figure 3.7b). When there indeed are options avail-

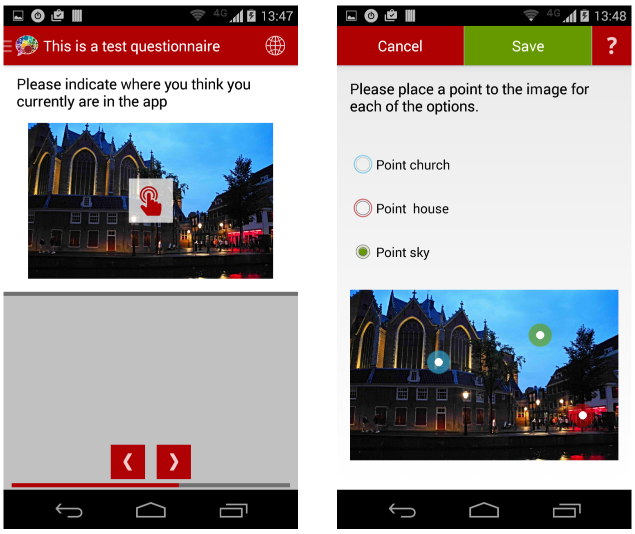


Please indicate where you would like to be given the aspects below

(a) Main layout (b) Detail

Figure 3.6: Tripole question screens

able, the user needs to select option he wants to indicate in the image and then touch the image at place he wants to relate the option to. If the designer did not specify additional options, the user should still place a point in the image but does not need to select an option first. The point then reflects the question itself.



Please indicate where you think you currently are in the app

Please place a point to the image for each of the options.

(a) Main layout (b) Detail

Figure 3.7: Clickable image question screens

3.5.4 Choice question

Choice questions do not have any detail screens and can be answered directly from within the questions list page. These questions can be configured to allow an ”other” option, which is then displayed together with a text field where user can specify the other option that they are missing. Furthermore, multiple choice selection questions can be configured to enforce a maximum number of options to select.

Chapter 4

Realisation

After analyzing the problem domain, specifying the requirements, designing the user interface and several meetings with the stakeholders I advanced to the implementation stage. Since the requirements were mostly known at the beginning, the Waterfall development process could be used, although some functionalities were reevaluated and changed during the development. Firstly a mockup prototype was created using FluidUI, which also simulated the navigation between the screens. Then I tried to get familiar with several Android functionalities I wanted to use such as the navigation drawer, SQLite databases, HTTP connections, and so on. For that reason, several minimalist prototype applications were created so I could get acquainted with yet unknown elements. After that, an implementation of the actual application begun, and a month later a first proprietary version was presented to the stakeholders.

4.1 Mockup prototype

In previous chapters, one could read that for the wireframes design I used a web application called FluidUI. Interestingly, FluidUI does not only allow to design static wireframes, but also enables its users to create an interactive prototype of their designed application. All one needs to do is to select elements like a menu item or a button in the wireframes and link them to another wireframe as an "onClick" action. Running the prototype allows to get a good feeling about the application, almost as if it was already implemented. One can test how the user interface feels, without writing a single line of code.

4.2 Packages

The application code is divided into several logic packages. The idea was to logically bind elements with similar functionalities together, so it is easy to navigate around the code for developers. They were created during the development process whenever it was effective. Finally, the application consisted of the following packages:

- **adapters** On Android, *Adapters* act as a bridge between data and a list view. They are bound to a list view and their task is to create or recycle rows for each data item. This package contains Adapters for all lists in the application including surveys list and questions list.
- **conf** This package contains a few configuration classes of fragments, languages and general configuration (e.g. mobile data file limit or file split size).
- **databases** All code, which works with any database is stored in this package. The databases are accessed through *DatabaseHelpers* wrapping the SQL queries by methods.
- **fragments** Fragments (which we could call "pages" for readers that are not familiar with Android development) are stored in this package.
- **helpers** This package contains classes used for text formatting, converting or other help functions used through-out the application.
- **models** This package contains all models used in the application. These include model of Survey, Media response and also all question types.
- **network** This package contains tasks used for network communication including *DownloadSurveyTask* or *SubmitParticipationTask*.
- **receivers** This package has only two classes which are called by the system when the device gets connected to wifi, or after specified amount of time. It is used for invoking the automated upload of responses.
- **tasks** This package contains background tasks for asynchronous operations (see paragraph 4.3).
- **views** Some *Views* had to be adjusted by inheriting from generic Android view classes. For example, the ClickableImageView and TripoleView are placed here.

4.3 Asynchronous operations

While developing almost any application, a developer sooner or later encounters a need to use asynchronous operations. On Android, application initially run on the main thread (also called the UI thread) which is shared among all application's components. This is the only thread that is allowed to draw or modify any visible user interface elements. Therefore, when a developer wants to access some data over a network, or load an image from the device, he should avoid using the UI thread, because that would freeze the whole device. Eventually, the application would even be killed by the system. Instead, the application must start an asynchronous operation in a separate thread, which loads the image, or downloads the data from Internet without blocking the main UI thread. However, when the data is ready, the separate thread cannot touch any user interface elements but must send the data to main thread instead. Subsequently, the main thread is responsible for updating the user interface. Since all this would be too annoying to implement every time we want to access data over Internet or do any other asynchronous operation, Android has a special class called AsyncTask[19], which keeps all this under the hood, while the programmer is asked to only implement methods *onPreExecute(), doInBackground() and onPostExecute().*

4.3.1 AsyncTasks

All AsyncTasks can be found in the *tasks* package or in the *network* package. The *network* package contains all tasks for network communication: for downloading surveys and uploading responses. The *tasks* package contains one *AsyncTask* which encrypts response databases, splits large files and moves everything to a special folder for completed responses that are ready for upload. Then, when the time comes, the network package's task *SubmitParticipationTask* goes through this directory and uploads all files it contains.

4.3.2 Image loading and caching

Another case that required asynchronous operations, was the decoding of image bitmaps. It turned out that decoding larger images could cause freezes of about 1000ms, which, of course, did not go unnoticed when changing a questions page. Therefore, an ImageLoader class was implemented in the support package that includes an *AsyncTask* inside. It accepts an *ImageView* and a file path or a byte array of the image as arguments, and updates the *ImageView* after decoding the image off the UI thread. Until the image is decoded and displayed, a loading icon is shown instead. In addition, an image cache was also implemented in order to prevent repetitive decoding of the same image in a short period of time. As a key to the cache I used either the file's path (for images added by the user as a response) or the survey, question and image id (for question images).

4.4 Question logic

Question logic (requirement F5) allows a creator of a survey to add rules, which will hide or show a question based on respondent's previous answer. Implementing question logic was one of the most challenging tasks, together with the Tripole and Clickable image questions. Every question, which has some logic rule assigned, needs to be aware of previous responses and display or hide itself when necessary. This must, of course, also work when two questions are on the same page and the second question is controlled by the previous one. Question logic was limited to choice question types as controlling questions. When a question has one or more rules assigned, it will only be shown when at least one of its rules is met.

When the controller and controlled questions are on separate pages, the implementation of logic rules is quite straight forward; just before the controlled question is about to be displayed, the application will first check user's answer and display or hide the question based on that. However, when controller and controlled questions are on the same page, things get more complicated. The controlled question needs to be notified about response changes of the controller question in order to allow it to react appropriately. This is accomplished by maintaining a list of controlled questions and sending them a change message. Because questions are loaded in the order they are displayed, the controlled question upon its creation locates its controller question and registers as a *Listener* for response changes. When user changes a response to the controller question, the *Fragment* calls *onAnswerChanged()* on the respective question and that question immediately calls onControlQuestionAnswerChanged() for each of the dependant (controlled) questions. Each dependant question can then react by hiding or showing itself and passing the call on to its own dependant questions, if any. It is important to add, that one question might even be controlled by multiple questions. In order to address this problem, each controlled question has a list of questions that make them visible. If this list is empty, the question is hidden and if not, the question is visible. A simplified sequence diagram of one page question logic is shown on figure D.2.

4.5 Option randomization

Questions with options can be designed in a way that requires randomization of its options' order (F17). Implementation was achieved by using Java's *Collections.shuffle()* method. However, in order to keep same order of options in one participation but different across multiple participations, an additional step must have been made. At the time a new participation is started, the application generates random seed number which is saved into *Participation Database*. Then, every time a question with randomized options is shown, the saved seed number is passed as a second argument to the *shuffle()* method, ensuring that the randomized options order is always the same in that participation even if it is closed and opened later. Additionally, when a participation is started, the device's location and time are also stored (F16).

4.6 Tripole and Clickable image views

The Tripole and Clickable image questions are unusual question types and required a special approach. For each of them special classes *TripoleView* and *ClickableImageView* extending *View* class were created. These views handle all drawing and touch events and can be used just as any other view in xml layout. The overridden method *onMeasure()* in TripoleView makes sure that the triangle is always equilateral.

Because the position of options in the Tripole can be designed to be random, it was necessary to save the distances from the touch point to each option (vertex) of the Tripole instead of simply saving the coordinates of the touch point. When the Tripole's answer is about to be displayed, the touch point is calculated by the distance of two options (vertices).

The Clickable image touch point saves coordinates, since the image does not allow randomization; i.e. it cannot be rotated and always has the same orientation.

4.7 Translations

The application must support multiple languages and allow users to easily switch between them. On Android, adding resources in multiple languages is easy. All resources are stored in xml files in specific directories containing the name of the language. Also, the Android system handles loading of the correct language resources itself. The only responsibility of the developer is to call a method that sets the respective language. After that, all newly created views will use the new resource. If the language changes, but some view is not recreated (such as navigation drawer in this case), the developer must request recreation of such a view in order to adapt to the new language setting.

4.7.1 Translations management

Still, managing translations for over 20 languages quickly becomes a complex and annoying task. This is because every time a new text resource is added to the resource, it also needs to be translated and added into all supported translation files. To ease this process, a Google spreadsheet that was shared with the whole team was created. This spreadsheet contains a sheet for each of the supported languages with 4 columns (Explanation, StringName, English and a column for the translation). The first three columns are linked to a sheet called 'Original English', where the terms are added, one per row. If one wants add or modify a translation for some language, he just needs to open the respective sheet and fill in the translation column. Furthermore, if new terms for translations are added into Original English sheet, they immediately appear as untranslated terms in all other language sheets. Last but not least, I have also created a Google spreadsheet script, which upon language selection generates a xml file, that can easily be added to the Android application every time new translations are provided.

4.8 Deployment

The final step in a software development is its publication, also called deployment. On Android, applications are usually published and distributed over Google Play Store. In order to do that, an application's profile must be created on the Play Store first. Then the application must be digitally signed with a developer's certificate and uploaded to the Play Store through Developer's console. It is important to note that once an application is signed and uploaded to Google Play Store for the first time, then all its future updates must be signed with the very same certificate. That is to ensure that future versions come from trustworthy origin and that no intruder can upload their own version of the application with possible malware in case the developer's account is compromised.

4.9 Implementation evaluation

Overall the implementation is considered as successful. The application performs flawlessly, is responsive, reliable, does not crash, and communicates smoothly with the server.

Unfortunately, the data encryption on the device was not implemented in the way it was planned. As was described in chapter 2.9.2, I wanted to use synchronous AES cipher to encrypt files with a key, that would be asynchronously encrypted. Naturally, this encryption puts a requirement on the server's backend, which needs to implement a corresponding decryption process. However, at the time of implementation, the backend's decryption process was not ready so this encryption method could not be used. Therefore, more simpler encryption method which is already supported by the server has been implemented, but the encryption method itself cannot not be discussed in the thesis because its security is based on secrecy and the stakeholder does not want to reveal these details.

If someone would implement the same application again, I would recommend to use a special library for network communications. Fortunately, network communication is not extensively used in this application so it is not a big issue. Nevertheless, after working with Retrofit [20] library for REST API in other projects I would not use plain *AsyncTasks* for network communications anymore.

4.10 Further development

The application does not need any additional implementation and is ready to be published. Even though there are already few requests for minor improvements, none of them involves an important feature or fix which needs to be included in the first public release. However, the whole Sprockler project is still waiting for its public launch because other necessary tools such as surveys designer or responses visualizer need to be finished.

For future versions and iterations of the development I would implement a compression algorithm for videos before their upload. Sadly, so far only commercial libraries to compress video files on Android were found.

Chapter 5

Testing

5.1 Testing by a programmer

Testing was done during the whole development after implementing every new functionality or screen. It is always better to find a bug sooner, when you are still well aware of the problem domain and context, than later. I appreciate the development and debugging tools that the Android platform provides. They were quite helpful when something was not working properly or behaved weird. Especially the detailed and visually attractive profiler or explorer of hierarchy views saved a lot of time.

5.2 Testing by testers

The Sprockler project team helped out in testing the software and providing quick feedback. This speedy testing was quite helpful to fix bugs promptly. To make the distribution of new application versions easier, a Google group for beta testers has been created and the application has been released on Google Play in beta mode. There were about three people actively testing the application after every release, and throughout the whole development process a total of thirteen people joined the beta group and tested the application. Most bugs or problems were usually reported within a few days. The rapid feedback was very valuable and helpful. Having all coding still clear in my mind, I could trace and solve most bugs rather quickly.

5.2.1 Exception reporting

While distributing the application to beta testers over the Google Play store, it sometimes happened that the application crashed. To help detect the error, Android devices can send a report of the callstack and exception message to the developer's Google Play console. The only problem is that this must be approved by the user and not all of them always do so. Furthermore, for yet unknown reasons, it usually takes several hours to receive such a report, leaving the developer in doubt in the mean time. Accordingly, fixes may take significantly longer than needed. After a while I started looking for alternatives and found Crashlytics [21] crash reporting platform. It does not only send the reports immediately, but also provides more detailed information of the crash and of all threads that your application was running. There is also *NewRelic* applications analysis platform that provides even more information about application's response times, usage and so on, but offers only paid services. Yet, I was absolutely satisfied with using Crashlytics.

5.2.2 Issue tracking

Having dedicated testers is always a great help for a developer, however, it can easily become difficult to keep track of all problems. This project was no exception. It quickly turned out that reporting issues via email is not convenient. Since I was already using a Bitbucket[22] as a git[23] repository and its ticketing system for developing tasks, I have asked the testers to join the ticketing system and submit all issues there. I could then more easily respond to the problems and avoid growing mail conversations covering several issues. The data from the Bitbucket's tracking system are exported and attached on the enclosed CD in a dedicated folder. I have also tried to export a mail communication as an illustration of the testing and reports from testers, but because of the complex conversations covering all kinds of issues and subjects, it appeared impossible (which just supports the usage of issue tracking).

Conclusions

In the beginning this thesis has briefly introduced and analyzed mobile applications of various survey platforms and evaluated them. Next, based on those findings and with accordance with the stakeholders' wishes, requirements for a new mobile survey application were set. After that, a graphical interface of the new application was designed, followed by a technical design of the application. Then, the implementation phase started after which testing and debugging begun.

The goal of this thesis was to develop a complete, reliable and functional application that could be used to interview people offline using mobile devices as part of new inquiry platform called Sprockler. All in accordance with the stakeholders' needs. Throughout the whole development process I have been in a close contact with the stakeholders. This assured full satisfaction with the result. The resulting application meets all the requirements and was appreciated and accepted by the stakeholders.

Although the application was also tested by the developer, it was extensively tested by the stakeholders' team and debugged accordingly. So far, it has been working flawlessly to the best of my knowledge. Since the application is a part of a new interviewing platform, some functionalities cannot be used or demonstrated to its full potential yet. This is due to limitations of other elements in the larger project, such as the server or the visualization of responses.

I think it is safe to say, that with the deliverable of a well working Android application, the project has been completed successfully.

Contribution

The application has already been used for collecting responses of inquiries that ran in Jerusalem and Brazil and more projects are about to come. I believe that it has proven itself as a reliable and very useful part of the survey platform and that it will be a great asset to the Sprockler project.

Personal evaluation

Thanks to this thesis I had a chance to get experience in developing a real product for a real stakeholders with all what it takes. It has been like no other school project and I learned many soft skills of software development. I had to go through the full software development process, starting with effort and time estimation and analysis of the stakeholders' wishes, over software and graphical design, to implementation and final testing. Furthermore, I also improved my project planning skills since I regularly had to provide a reasonable time plan and then had to make sure I delivered as scheduled. Additionally, I believe I also enhanced my communication skills since I had to extensively communicate and discuss all things with the stakeholders in English.

However, I did not only improve my software engineering skills, but also learned many new things about the Android platform and have also gained some experience in designing a user interface.

Lastly, it has been a great experience to stay in the Netherlands and explore its beautiful parts while being there at the beginning of the development period, and when travelling for meetings. I enjoyed working on this application and thesis from the beginning to the end and I am very glad I had this opportunity.

Bibliography

[1] Cognitive Edge Pte Ltd. Sensemaker homepage. [Online, accessed: 2015-01-24]. Available from: `http://www.sensemaker-suite.com/`

[2] Cognitive Edge Pte Ltd. Cognitive Edge homepage. [Online, accessed: 2015-01-24]. Available from: `http://cognitive-edge.com/`

[3] Cognitive Edge Pte Ltd. Sensemaker Android application. [Online, accessed: 2015-02-15]. Available from: `https://play.google.com/store/apps/details?id=com.cognitiveedge`

[4] The Apache Software Foundation. Cordova project. [Online, accessed: 2015-02-24]. Available from: `http://cordova.apache.org/`

[5] SurveyMonkey Ltd. SurveyMonkey - About us. [Online, accessed: 2015-02-16]. Available from: `https://www.surveymonkey.com/mp/aboutus/`

[6] SurveyMonkey Ltd. The SurveyMonkey App for iPhone® and iPad®. [Online, accessed: 2015-02-16]. Available from: `https://www.surveymonkey.com/mp/iphone-survey-app/`

[7] Google Inc. *Android Design*. [Online, accessed: 2015-01-24]. Available from: `https://developer.android.com/design/index.html`

[8] Google Inc. Platform Versions. [Online, accessed: 2015-03-09]. Available from: `https://developer.android.com/about/dashboards/index.html#Platform`

[9] Adobe Systems Inc. Adobe Announces Agreement to Acquire Nitobi, Creator of PhoneGap. [Online, accessed: 2015-03-08]. Available from: `http://www.adobe.com/aboutadobe/pressroom/pressreleases/201110/AdobeAcquiresNitobi.html`

[10] Anderson, T. Microsoft, Xamarin give Visual Studio a leg-up for... iOS and Android? [Online, accessed: 2015-03-08]. Available from: `http://www.theregister.co.uk/2013/11/13/microsoft_xamarin_ios_android_c_sharp_visual_studio/`

[11] Binstock, A. .NET Alternative In Transition. [Online, accessed: 2015-03-08]. Available from: `http://www.informationweek.com/architecture/net-alternative-in-transition/d/d-id/1098050?`

[12] Xamarin Inc. Xamarin pricing. [Online, accessed: 2015-03-09]. Available from: `https://store.xamarin.com/`

[13] Google Inc. Introduction to Android. [Online, accessed: 2015-03-09]. Available from: `https://developer.android.com/guide/index.html`

[14] Google Inc. SQLite Database. [Online, accessed: 2015-03-10]. Available from: `http://developer.android.com/reference/android/database/sqlite/SQLiteDatabase.html`

[15] Reenskaug, T.; Coplien, J. O. The DCI Architecture: A New Vision of Object-Oriented Programming. [Online, accessed: 2015-04-12]. Available from: `http://www.artima.com/articles/dci_vision.html`

[16] Gamma, E.; Helm, R.; Johnson, R.; et al. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1994.

[17] Fluid Software Ltd. Fluid UI. [Online, accessed: 2015-03-25]. Available from: `https://www.fluidui.com`

[18] Google Inc. Android Support Library. [Online, accessed: 2015-04-13]. Available from: `http://developer.android.com/tools/support-library/index.html`

[19] Google Inc. Android AsyncTask. [Online, accessed: 2015-03-25]. Available from: `http://developer.android.com/reference/android/os/AsyncTask.html`

[20] Square, Inc. Retrofit. [Online, accessed: 2015-04-02]. Available from: `http://square.github.io/retrofit/`

[21] Twitter Inc. Crashlytics. [Online, accessed: 2015-03-28]. Available from: `https://crashlytics.com/`

[22] Atlassian Pty Ltd. Bitbucket. [Online, accessed: 2015-04-02]. Available from: `https://bitbucket.org/`

[23] L. Torvalds and others. Git. [Online, accessed: 2015-04-10]. Available from: `http://git-scm.com/`

Appendix **A**

Acronyms

AES Advanced Encryption Standard

API Application Programming Interface

GUI Graphical User Interface

GPS Global Positioning System

HTTP Hypertext Transfer Protocol

HTTPS Hypertext Transfer Protocol Secure

IDE Integrated development environment

JSON JavaScript Object Notation

REST Hypertext Transfer Protocol

SDK Software Development Kit

SPA Single Page Application

UC Use Case

UI User Interface

XML Extensible Markup Language

Appendix **B**

Contents of enclosed CD

```
readme.txt ....................... the file with CD contents description
src ..................................... the directory of source codes
  thesis ............. the directory of LaTeX source codes of the thesis
text
  thesis.pdf .......................... the thesis text in PDF format
issueTracking ........................... the directory of issue tickets
```

Appendix C

Development tools

C.1 Android studio

Android studio is by Google the promoted IDE for Android. It is based and built on top of IntelliJ IDEA software, which is one of the most popular and advanced integrated developing environments. Eclipse with Android development plugin was the standard until the release of Android studio in May 2013. Since then, developers started to move to Android Studio. Android studio is far superior to Eclipse and offers many new and improved features.

One of those new features is integration of Gradle. Gradle is a tool for build automation and, apart other useful features, allows developers to easily manage dependencies by adding only one line for each dependency. Furthermore, it allows developers to smoothly create and manage build configurations. For this project build configurations for a release (production) build, Beta build and a Debug build were created. The Production build connects to a production server, while Beta build uses test server and Debug build connects to a local server. Additionally, the Beta and Production builds use ProGuard to shrink, optimize, and obfuscate the code in order to make reverse engineering more difficult. Then, at the time of build, the developer only needs to select what type of build configuration he wants to run without changing anything in the code itself.

C.2 Git

Using Git or any other versioning system is a common practice and almost a must in a software development. Although it is especially useful for projects consisting of multiple developers, even projects with one developer can find versioning systems convenient; especially for reverting already saved changes. This project's repository was hosted at bitbucket.org [22].

C.3 Issue tracking

To keep a track of all tasks and issues during the project a ticketing system which came with the git repository at bitbucket.org [22] was used. Furthermore, once testers joined the team they were asked to use the issue tracking system as a reporting platform for submitting bugs and encountered issues (more in the section 5.2.2).

Appendix D

UML Diagrams

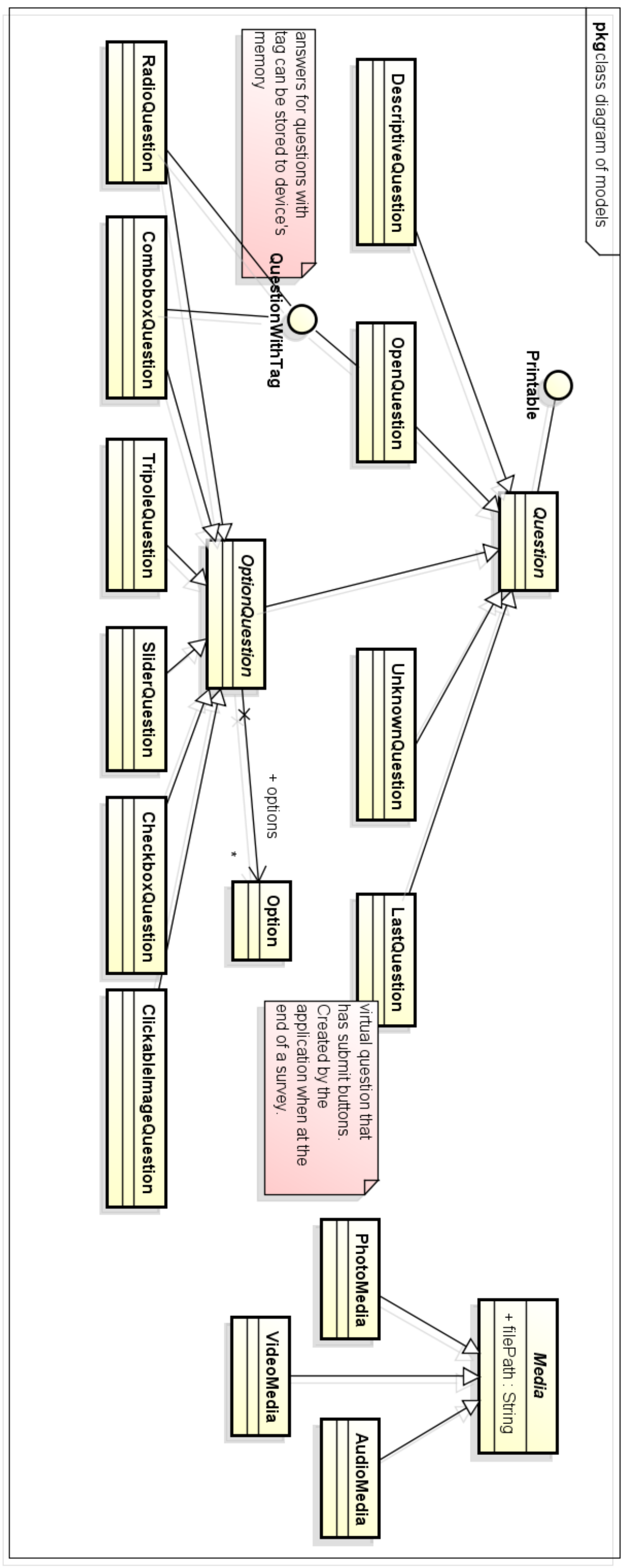


Figure D.1: Class diagram of questions

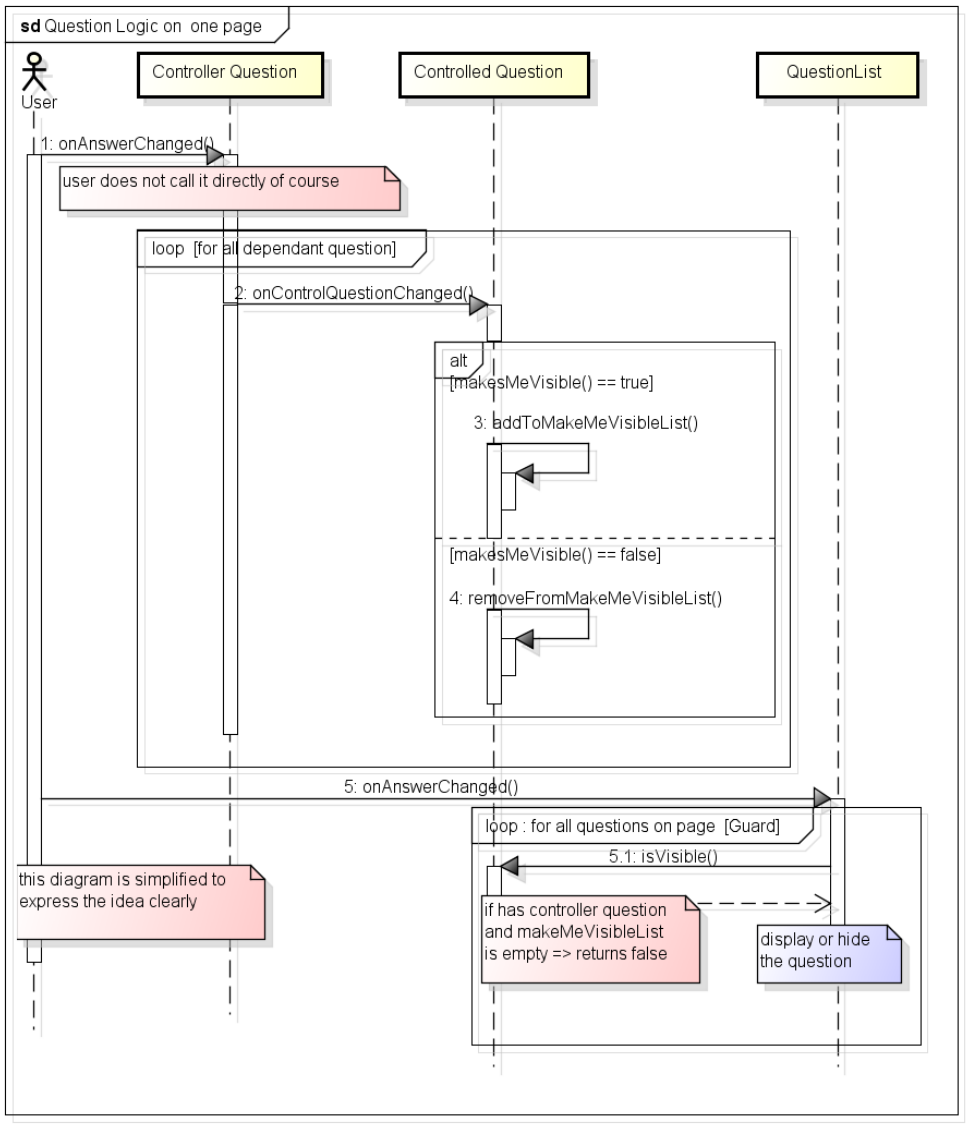

sd Question Logic on one page
User
Controller Question
Controlled Question
QuestionList
1: onAnswerChanged()
user does not call it directly of course
loop [for all dependant question]
2: onControlQuestionChanged()
alt
[makesMeVisible() == true]
3: addToMakeMeVisibleList()
[makesMeVisible() == false]
4: removeFromMakeMeVisibleList()
5: onAnswerChanged()
loop : for all questions on page [Guard]
5.1: isVisible()
this diagram is simplified to express the idea clearly
if has controller question and makeMeVisibleList is empty => returns false
display or hide the question

Figure D.2: Sequence diagram of question logic on one page

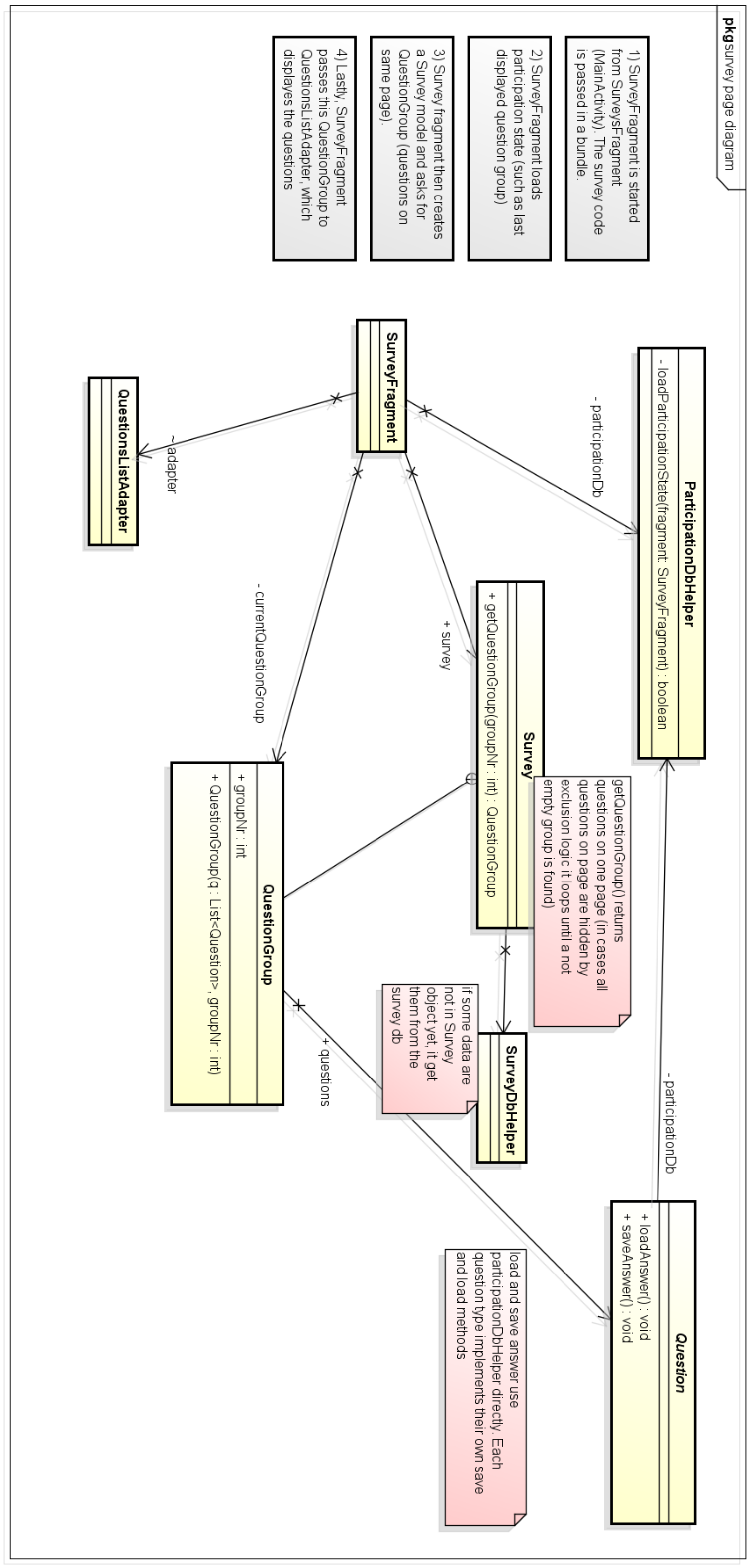

pkg survey page diagram
1) SurveyFragment is started from SurveysFragment (MainActivity). The survey code is passed in a bundle.
2) SurveyFragment loads participation state (such as last displayed question group)
3) Survey fragment then creates a Survey model and asks for QuestionGroup (questions on same page).
4) Lastly, SurveyFragment passes this QuestionGroup to QuestionsListAdapter, which displayes the questions
ParticipationDbHelper
- loadParticipationState(fragment: SurveyFragment) : boolean
- participationDb
SurveyFragment
QuestionsListAdapter
~adapter
Survey
+ getQuestionGroup(groupNr : int) : QuestionGroup
+ survey
- currentQuestionGroup
getQuestionGroup() returns questions on one page (in cases all questions on page are hidden by exclusion logic it loops until a not empty group is found)
SurveyDbHelper
if some data are not in Survey object yet, it get them from the survey db
QuestionGroup
+ groupNr : int
+ QuestionGroup(q : List<Question>, groupNr : int)
+ questions
- participationDb
Question
+ loadAnswer() : void
+ saveAnswer() : void
load and save answer use participationDbHelper directly. Each question type implements their own save and load methods

Figure D.3: Survey page class diagram

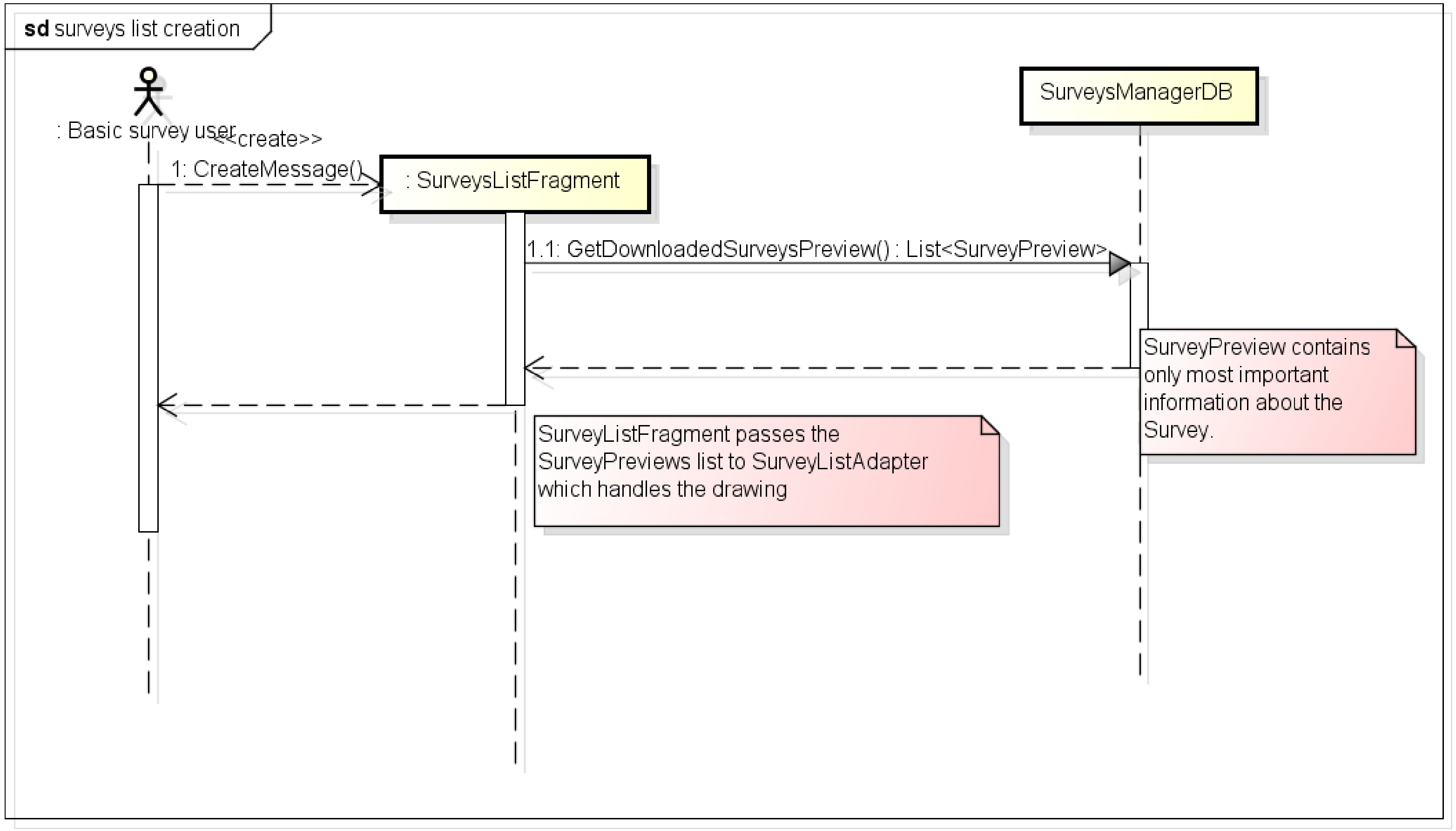

sd surveys list creation
: Basic survey user
<<create>>
1: CreateMessage()
: SurveysListFragment
SurveysManagerDB
1.1: GetDownloadedSurveysPreview() : List<SurveyPreview>
SurveyPreview contains only most important information about the Survey.
SurveyListFragment passes the SurveyPreviews list to SurveyListAdapter which handles the drawing

Figure D.4: Surveys list loading sequence diagram