



---

<sup>b</sup>  
**UNIVERSITÄT  
BERN**

# **Zeeguu Translate Application**

**Extending the Zeeguu Platform to the Android Device**

## **Bachelor Thesis**

Pascal Giehl

from

4663 Aarburg, Switzerland

Faculty of Science

University of Bern

August 19, 2015

Prof. Dr. Oscar Nierstrasz

Dr. Mircea Lungu

Software Composition Group

Institute of Computer Science

University of Bern, Switzerland

# Acknowledgements

First I would like to thank *Prof. Dr. Oscar Nierstrasz* for the opportunity to do my bachelor thesis in the Software Composition Group of the University of Bern.

Furthermore I would like to express my appreciation to *Dr. Mircea Lungu* who invested a lot of effort to support me and the project. You really feel the passion and creativity he got when working on such projects. Thanks also for many inspiring and fruitful discussions about new ideas and concepts.

Another thank you I would like to state to the Zeeguu team consisting of *Karan Sethi*, *Linus Schwab* and *Ada Lungu* who improved the Zeeguu platform during the same time as I did. They supported the applications by adding new APIs to make new features available, sharing code in a common library or improving the web interface for the browser to match the mobile version.

Additionally I appreciate that many people have taken the time to assist this project behind the scenes with their advice, knowledge and time.

And finally I would like to express my thanks to all the people who took the time and patience to join the usability tests.

# Abstract

Zeeguu is a user based learning platform which enables the user to expand his vocabulary and receive feedback about his progress while he is reading about topics he loves.

To reach the best efficiency possible, the user should be able to access the Platform from as many devices as possible, all connected with one single account. This way Zeeguu supports the user when he is reading foreign languages, monitors his progress of those languages and gives him further possibilities to improve the language everywhere and at any time.

This paper introduces an Android application as an extension for the Zeeguu ecosystem and since Android is the leading mobile operating system, this work represents an important step towards getting Zeeguu into the daily life of the users.

# Contents

<b>1</b>	<b>Introduction</b>	<b>5</b>
<b>2</b>	<b>Related Work</b>	<b>6</b>
2.1	Translation Platforms . . . . .	6
2.1.1	Google Translate . . . . .	6
2.1.2	Bing Translator . . . . .	7
2.1.3	Amazon Kindle Translate . . . . .	7
2.1.4	iTranslate . . . . .	7
2.2	Online Learning Platforms . . . . .	7
2.2.1	Babbel . . . . .	7
2.2.2	Duolingo . . . . .	8
2.3	The Zeeguu Platform . . . . .	8
2.3.1	The Zeeguu Chrome Plug-in . . . . .	8
2.3.1.1	Zeeguu Quantifier . . . . .	9
2.3.1.2	Zeeguu RSS Reader . . . . .	9
<b>3</b>	<b>The Challenge</b>	<b>10</b>
<b>4</b>	<b>The Solution</b>	<b>12</b>
4.1	The Idea . . . . .	12
4.2	Preliminary Feasibility Study . . . . .	12
4.2.1	Choice of the Mobile Operating System . . . . .	12
4.2.2	The Scope of the Application . . . . .	14
4.2.3	Modes Of Input . . . . .	14
4.2.3.1	Voice Recognition . . . . .	15
4.2.3.2	Optical Character Recognition . . . . .	15
4.3	The Architecture of the Zeeguu Android App . . . . .	17
4.3.1	The Front-End . . . . .	17
4.3.1.1	The Main Application Activity . . . . .	18
4.3.1.2	The Fragments . . . . .	19
4.3.1.3	Dialogs . . . . .	22

<i>CONTENTS</i>	4
4.3.2 Back-End . . . . .	23
4.3.2.1 Using the Zeeguu Server API . . . . .	23
4.3.2.2 The Interface to the Zeeguu API . . . . .	24
<b>5 The Validation</b>	<b>26</b>
5.1 Zeeguu Application Goal . . . . .	26
5.2 Usability tests . . . . .	27
5.2.1 Methodology . . . . .	27
5.2.2 Results . . . . .	28
<b>6 Conclusion and Future Work</b>	<b>29</b>
6.1 Conclusion . . . . .	29
6.2 Future Work . . . . .	30
6.2.1 Diversify into iOS and Windows Mobile . . . . .	30
6.2.2 Improving the Zeeguu Android Application . . . . .	30
6.2.2.1 Different Exercises . . . . .	30
6.2.2.2 Display Statistics and User Information . . . . .	31
<b>A Anleitung zu Wissenschaftlichen Arbeiten</b>	<b>32</b>
A.1 Support new Languages . . . . .	32
A.1.1 Modify the Language Arrays . . . . .	33
A.1.2 Implement all Flags . . . . .	33
A.2 Manage the Sliding Menu . . . . .	35
A.2.1 Create a new Fragment . . . . .	35
A.2.2 Add a Fragment . . . . .	35
A.2.3 Remove a Fragment . . . . .	37
A.3 Create a new Dialog . . . . .	38
A.4 Fragment Communication . . . . .	38
A.5 Add new API requests . . . . .	39
<b>B Designing User Interfaces on Android</b>	<b>44</b>
B.1 Activity . . . . .	44
B.2 Fragment . . . . .	45
B.3 Dialog . . . . .	45
B.4 Interfaces . . . . .	45
<b>C Usability Testing</b>	<b>47</b>
C.1 Initial Situation . . . . .	47
C.2 Test Results . . . . .	48

# 1

## Introduction

Professor Stephen Krashen illustrated the importance of free reading to learn new languages. This approach – in schools referred to as *sustained silent reading* or SSR – leads to increased motivation, which in turn results in higher time investment, and a stronger focus on the material read which increases the retention of the learned words.

Zeeguu’s main idea is to provide users who learn new languages with the many advantages of the free reading experience. Evolving from this idea, Simon Marti and Dr. Mircea Lungu started working on Zeeguu, a web-based platform that allows the user to get instant translations if a word is unknown while he browses “freely through the web”[2]. Zeeguu then keeps track of the words that have been looked up and offers exercises to memorize these words.

This thesis will introduce the Zeeguu Android application, which extends the Zeeguu ecosystem. It allows the user to have Zeeguu within his reach wherever he goes and whatever he does by supporting a device that most people these days always have within their reach: their smartphone.

The main goal of the Zeeguu Android application is to support the user in accessing the Zeeguu platform ubiquitously. Not only will this lead to a higher turnover of translated words, but it could boost the time invested by the user into the Zeeguu training exercises which in turn will result in a faster acquisition of the vocabulary of the new language.

Therefore the main technical contribution of this thesis is the Android application that is going to fundamentally change the way the Zeeguu platform is used today.

# 2

## Related Work

This chapter will provide a short overview of related work, both work that is still in progress and work that has resulted in products already available on the market.

### 2.1 Translation Platforms

A translation platform provides a translation of a word or a sentence from one language to another. It does not necessarily contain functions that would support a user to learn a new language. The goal of such a platform is only to support the user if he needs to translate a text. Popular platforms are:

#### 2.1.1 Google Translate

Google Translate (<https://translate.google.com>) is a translation service provided by Google. It is one of the biggest translation services with 90 languages supported and is used by 200 Million [4] people every day. It provides translations for single words as well as translations for entire sentences, documents or webpages.

The translator is also available on the most popular smartphones like Android, iPhone and Windows Phone. The platform allows users furthermore to save translations on the website as well as on the phone, but it does not provide any exercises based on those words that the user defined as important.

The API of this translation service allows external programs to access and use it for a usage fee.

### 2.1.2 Bing Translator

Bing Translator (<https://www.bing.com/translator>) is the translation service from Microsoft and is one of the biggest competitors of the Google translation service. It currently supports 51 languages and is also capable of translating from single words up to entire documents and webpages.

A mobile version is only available for Windows Phone users and the platform neither allows users to save translations on the phone nor does it provide exercises for the user.

The API also allows external programs to use the service to translate words or sentences and provides up to 2 million characters a month for free with the option to raise the limit with a monthly subscription fee.

### 2.1.3 Amazon Kindle Translate

Kindle (<https://www.amazon.com/gp/digital/fiona/kcp-landing-page>) is an application provided by Amazon that enables users to buy, download and read e-books, newspapers, magazines or other digital media. The application also provides instant translation while reading any media on the platform, but it only works within the application. Furthermore it does not support any other features which help the user to acquire new languages like for example exercises.

### 2.1.4 iTranslate

The free application called iTranslate (<http://www.itranslateapp.com>) supports 90 different languages and is one of the most popular translators on most of the mobile devices. It allows you to translate words and sentences on your mobile device and keeps track of what you searched. It is only available on mobile phones and does not provide additional support to learn any languages.

## 2.2 Online Learning Platforms

Online learning platforms provide specific training possibilities and exercises which can be accessed through the web browser and often also through other devices. The goal of those platforms is not to provide translations, but to support users to increase their vocabulary and language understanding.

### 2.2.1 Babbel

Babbel (<http://www.babbel.com>) is a fee-based online platform that currently offers 14 languages and has over 20 million users. Its concept is to enable users to learn new



Figure 2.1: Zeeguu plug-in example

languages interactively through daily exercises until a certain level. But Babbel does not allow users to learn languages through reading media as it does not provide solutions for advanced learners. The Babbel ecosystem is accessible from a computer as well as on most of the mobile platforms.

### 2.2.2 Duolingo

Duolingo (<https://www.duolingo.com>) is a free online platform that supports 22 languages and has approximately 60 million users out of which about 20 million of them are active. Its concept is pretty similar to Babbel and the platform has also about the same disadvantages as Babbel.

## 2.3 The Zeeguu Platform

Zeeguu is the main platform that the Zeeguu Android application will use and was created in 2013 as a bachelor project from Simon Marti [2]. It is currently running on the server of the University of Bern. It provides an API to translate words and sentences and registers the words that were searched and bookmarked. Those words then serve as a basis for personalized exercises.

### 2.3.1 The Zeeguu Chrome Plug-in

The Chrome plug-in uses the Zeeguu API to allow the user to instantly translate words into his native language while he reads texts in other languages. The translation appears directly in the text itself as a popup and thus the user does not need to shift his focus away from the text (see Figure 2.3.1).

Next to this thesis, there are two other bachelor theses working on Zeeguu concurrently. One has the goal to improve the user model on the main platform so that Zeeguu is able to give smart feedback. The other one is another Android application which is going to be a RSS reader. Here is a brief overview of those two theses:

### **2.3.1.1 Zeeguu Quantifier**

Karan Sethi is currently working on the Zeeguu Quantifier thesis on a user model that will approximate how well the user knows the language. The data it uses therefore is collected from the translations that are made in combination with the context of those translations and the exercises that were passed or failed. This data will determine how likely it is that the user knows certain words. This can be used to provide the user with feedback on how well he knows the language and how fast he is progressing in learning it. In the future, the feedback from the Quantifier can be integrated in the Zeeguu Android application.

### **2.3.1.2 Zeeguu RSS Reader**

Linus Schwab is developing on a Zeeguu RSS Reader for Android. This thesis aims to bring the Zeeguu ecosystem also to those people who stay informed through RSS Feeds. Because RSS Readers allow readers to define what news and information they want to read, they can also perfectly tap into the SSR advantage and help users read articles that interest and fascinate them.

# 3

## The Challenge

Zeeguu grew out of a unique need that is not yet supported: providing the user with translations whenever and wherever he needs them so that he can benefit from the SSR experience while keeping track of all his progress and supporting him with individually tailored exercises.

As I showed in the related work none of the two parts of this concept are new, neither to provide translations nor to supply training exercises. But it is the combination of both parts integrated together that makes the idea different and, above all, unique.

Dr. Stephen Krashen[3] mentioned that short daily SSR sessions where students are allowed to freely choose and read books is a very successful way of acquiring a new language. But exactly this freedom is what Zeeguu partially lacks at the time of writing this thesis. The user can browse freely through the web with the Zeeguu plug-in, but he cannot read books, texts or any other material that is not inside the browser window.

The main problem is that the Zeeguu ecosystem is not available at places where the user does not have access to a computer and this lack of access to the platform lowers the chances of its success.

Furthermore a study published by Publishing Technology [5], which polled 3,000 consumers in the UK and US, showed that 44% of the smartphone owners said that they have read an eBook or part of an eBook in the last twelve months. Moreover 66% of respondents who mentioned they read eBooks on their smartphone said they did so more frequently today than they did a year ago. The result of this study demonstrates the potential of a new growing market that Zeeguu misses out yet completely.

Another proof of the importance of being present on the mobile platform for an educational approach is also the fact that many other online learning platforms provide

applications for the smartphone.

Thus the main challenge of this thesis is to develop a mobile application that expands the Zeeguu ecosystem. The application should enable the user to use most of the functions that are provided by the online platform on his mobile device. Thus he should be able to translate words that he does not understand on his device everywhere and at any time. Additionally there should be an option to bookmark those translated words so that they will be saved in his online profile.

Furthermore the application should display a list consisting of the words that the user bookmarked. This feature will allow the user to have an overview of all his learned words and he can easily recall them if he needs to.

The last user story that should be supported is the possibility to do exercises on the phone, which enables the user to train the words he bookmarked. This feature will guarantee that the user can train his vocabulary at any time. Thus for example if he needs to wait for some minutes he can use the time to improve his vocabulary on his mobile device.

In short a typical use case would be that the user did not understand one specific word when he was reading the newspaper in the morning and thus opens the application to translate and bookmark that word. After a while he cannot remember the word and so he opens the application again to search it in the list that consists of all the words he bookmarked. And finally in the evening when he is waiting at the bus stop he starts to do some exercises on the application to enhance his vocabulary.

# 4

## The Solution

### **4.1 The Idea**

The challenge of extending the Zeeguu ecosystem to increase its accessibility can be achieved through platform diversification: Zeeguu should support as many different platforms and devices as possible, so that it can be used by anybody everywhere.

The browser plug-in is responsible that every unknown word is highlighted when the user is on the computer, the mobile application would secure that Zeeguu is in reach when the user leaves his desk and the tablet version would provide an enjoyable big size version at home.

### **4.2 Preliminary Feasibility Study**

Before the first line of code can be written, many project related decisions need to be made. To guarantee a solid work a lot of research had to be done and this ended in a lot of reading and testing libraries.

#### **4.2.1 Choice of the Mobile Operating System**

There are many different mobile operating systems on the market and the question is inevitable which one would provide the best ground for the application to grow on. This decision will also define in which language the application will be written. To support multiple mobile operating systems, an application can either consist of multiple versions

written natively for each of the corresponding operating systems or it can be written in a common language and then automatically compiled for the individual platforms.

**Alternative 1: Code Generation-Based Solutions** To generate an application that runs on multiple platforms a programmer can use a programming platform that allows him to write code once and use code generation to create applications for different types of devices like Android, iPhone and the Windows Phone. Two of the most popular platforms right now are Appcelerator's Titanium<sup>1</sup> and Adobe's PhoneGap<sup>2</sup> which both use a combination of web languages combined with JavaScript. The big advantage of such an approach is that an application only needs to be created once for all platforms, which saves a lot of time. The disadvantage is that the application cannot access device related APIs and is not as dynamic and fast as a native application.

**Alternative 2: Native Code Solutions** The other alternative is to write the application for every single operating system from scratch. The application is then written in the native language of the operating system which means that it can access all the device related APIs. For example for Android Java would be used, for iOS Objective-C and for Windows Phone C Sharp. The advantages and disadvantages for this alternative are exactly the reverse of the previous alternative.

**Decision: Alternative 2** The goal was to extend the Zeeguu platform in a way that the application could target the largest market and at the same time would not have to compromise the user experience. This quality was finally only achievable through alternative 2.

To evaluate the largest market the numbers had to be analyzed and they clearly showed that the strongest member in the market is the Android operating system with a market share of around 80%<sup>[6]</sup> (2014 Q1: 81,2%, 2015 Q1: 78%) of all sold smartphones, which at the same time is the highest share ever reached in the global smartphone market<sup>3</sup>. Hence the decision was in favor for the Android operating system.

An additional reason, which backed the decision, was a new tool developed by Google called j2objc<sup>4</sup>, which allows the programmer to convert the logic of an Android application from Java to Objective C so that it could be used on an iPhone. This means that up to 70% of the code can be reused while only about 30% of the code has to be rewritten and the biggest part of those 30% is code that defines the GUI, which anyway has to be customized to meet the standards of the single operating systems. Many

---

<sup>1</sup> <http://www.appcelerator.com>

<sup>2</sup> <http://phonegap.com>

<sup>3</sup> [http://en.wikipedia.org/wiki/android\\_\(operating\\_system\)](http://en.wikipedia.org/wiki/android_(operating_system))

<sup>4</sup> <https://github.com/google/j2objc>

applications on the market already used j2objc successfully. Famous examples provided by Google itself are the Inbox or the Google Sheets<sup>5</sup> application.

This means that the weak spot of alternative 2 just got less severe and so the main advantages of this alternative were even more predominant. And if in a future work the application is transferred to iOS, Zeeguu would cover almost the entire market, because Apple's iOS market share is almost the missing 20%<sup>[6]</sup> (2014 Q1: 15,2%, 2015 Q1: 18,3%).

## 4.2.2 The Scope of the Application

Another question before the first line of code is written is to determine the scope of the application. We evaluated three alternatives here:

**Helper Application** A small helper application would support the main platform, but it would be at the same time also dependent on it.

**Extension Application** An extension application would extend the features that a platform provides.

**Standalone Application** If the scope defines a standalone version of the program, the application needs to provide all functionalities that the main platform provides without its reliance.

**Decision: Standalone Application** Because the main goal of the application was to guarantee the SSR advantages, it would need many key features of the platform integrated into the mobile program. Thus the decision was that a mobile standalone version was needed which could be used without the need to touch the Chrome browser at all.

Thus the scope was defined which meant that also the goals could be fixed. Those goals included to create a user management system, implement a translator that would also show the own wordlist, provide exercises for the user, synchronize everything with the server and support as many input modes as possible.

## 4.2.3 Modes Of Input

At the beginning of the thesis we envisioned supporting optical character recognition (OCR) and voice recognition (VR) next to the normal keyboard input. To implement such complex processes successfully, the application would need to use libraries that would allow us to transform inputs in form of a picture or a soundtrack into characters. The

---

<sup>5</sup><http://j2objc.org/docs/Projects-that-use-J2ObjC.html>

reason libraries should be used in the first place is that OCR and VR are very complex operations that need a lot of expertise next to many years of work to provide an acceptable recognition rate and this project lacks at least the time. Furthermore libraries are often still supported which means that they are still updated from time to time and therefore are able to provide state of the art algorithms.

But we need to test the libraries first before we implement them to learn whether they provide a good recognition rate, if the phone space they consume is within the limits and how fast they operate. For this reason the thesis required to create, use and customize many simple applications to be able to evaluate those libraries.

#### 4.2.3.1 Voice Recognition

Voice recognition is the process where an algorithm analyzes a spoken word or sentence and can convert it correctly into written characters. Google provides a voice recognition API<sup>6</sup> on Android phones for free that takes almost no space on the phone and has a good recognition rate in all languages. It is hardly beatable by any other library on the market and that is why the Zeeguu application uses it for its voice recognition.

#### 4.2.3.2 Optical Character Recognition

Optical character recognition is a process in which an algorithm analyzes a picture of a word or sentence and tries to recognize text inside it. If a word has been recognized, the library then returns the solution in form of characters.

The big question for this feature was if we process the OCR on the mobile phone itself or on the server. The decision was clearly in favor of the mobile phone because of the huge data traffic that had to be managed otherwise, the missing server infrastructure and the lack of resources to do such consuming calculations on the server.

To find a mobile OCR library that has a good recognition rate and a fast calculation time is very difficult. Even though there are a few that can be considered (*i.e.*, the ABBYY Mobile OCR Engine, Asprise Java OCR , OpenCV and Tesseract), most of them are not free which is the reason why they are not used for this project.

The best candidate that we elaborated was the Tesseract-OCR library which has an Android version called tess-two. It is a library originally created by HP and in the meantime sponsored by Google. It can be considered as one of the best open source OCR libraries.

But after many tests of the tess-two library for Android the measurements spoke a different language. The recognition rate was not as accurate as expected and often identified many very strange words when it had to recognize multiple words at once.

---

<sup>6</sup><https://www.google.com/intl/en/chrome/demos/speech.html>

Additionally the space it took on the phone was way beyond the planned value and would have increased the size of the application by many times. Those disadvantages would have resulted in a worse feeling of the overall application which would have lowered the chances of the application to be successful.

Thus after many trials and errors and even though the OCR function was one of our goals in the beginning we waived it. The gains to drop it were bigger than those to implement it.

## 4.3 The Architecture of the Zeeguu Android App

This section will explain the construction of the application by describing the most important classes and their relation to each other. The application itself can be divided into front- and back-end. The front-end handles all fragments and dialogs of the user interface whereas the back-end is responsible for the logic and the connection with the server.

But the line between front- and back-end can not be drawn perfectly clear in Android applications because activities and fragments provide both as they can manage a user interface and still execute logic methods in the background. But to simplify the architecture into one easy to understand picture the fragments are considered as front-end. The Figure 4.1 is a simplified architecture of the Zeeguu application<sup>7</sup>.

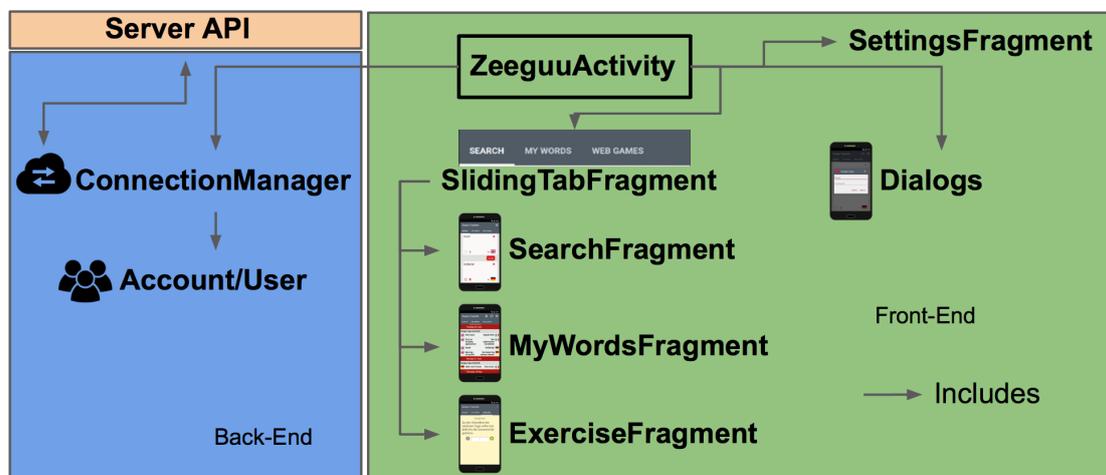


Figure 4.1: Zeeguu architecture

### 4.3.1 The Front-End

*“Any product that needs a manual to work is broken.” - Elon Musk*

Elon Musk, CEO and product architect of Tesla Motors, CEO and CTO of SpaceX and chairman of SolarCity, suggests that the user interface design is the key to success or failure of an application. Indeed, the user interface defines what the user thinks about the application. A program can have the best back-end or logic if its user interface is not intuitive and appealing it will fail to persuade the user in the end.

Since design is essential for success, Zeeguu aims to be intuitive and easy to handle.

<sup>7</sup>The source code for the application is open-source and can be found on GitHub [7]

### 4.3.1.1 The Main Application Activity

Every application needs at least one activity and often there is even the need for multiple ones. The Zeeguu application strived to possess only one single activity. All dynamic parts are created through changing fragments<sup>8</sup>. The main advantage of this architecture is that the application is faster than if multiple activities would have been used.

A fast application has one major advantage: It arouses the feeling of intuitiveness because the response after an action will almost be immediate.<sup>9</sup>

This single activity inside Zeeguu is the `ZeeguuActivity` class and it contains the `ConnectionManager`, which handles the link to the server. The `ConnectionManager` then can be accessed through a getter method in the activity that is implemented as part of an interface. An external class can now cast the activity into that interface and then can call this get method to receive the `ConnectionManager` from the activity. The principle of using interfaces for fragment communication is described in detail in section B.4 interfaces. The only difference is that the main activity does in this case not refer to another fragment, but instead just returns the `ConnectionManager`.

To guarantee this callback the `ZeeguuActivity` needs to implement that interface that accepts it and every fragment that uses those callbacks needs to cast the `ZeeguuActivity` into this interface.

Furthermore the `ZeeguuActivity` also saves a reference of all fragments that are used are used, so that they can be restored when the activity gets destroyed, for example if the phone is rotated.

Even though all fragments are referenced in the activity, only two fragments are managed by the activity itself while all other fragments will be handled through a separate fragment (see Figure 4.1). One of those two fragments is always active while the other is not. The active fragment defines the view and handles all the logic while the other just waits to be called again from the user.

1. One fragment that the `ZeeguuActivity` manages is the `SlidingTabFragment` which provides the main view that is most of the time active. This fragment puts other fragments into a sliding menu and shows always one at a time.
2. The other fragment is the `PreferenceFragment` which is responsible for displaying the preference screen. It lists all possible preferences that the user can change in the application and handles the requests the user entered.

---

<sup>8</sup>See section B.2 fragments.

<sup>9</sup>This might be related to the effect that Daniel Kahnemann [1] describes in his book *Thinking Fast and Slow*. He argues the longer a feedback takes after an action, the more repetitions and time humans need to learn it. For example, he argues that one reason why blackjack is easier to master than chess is that a person gets feedback faster in blackjack in form of the outcome of the game than he does in chess and so he can easier break down the single moves into good or bad ones. It is the same for computer programs: the faster an application reacts and therefore provides feedback, the faster and easier a user understands the behavior of the application and the more intuitive he rates it in the end.

### 4.3.1.2 The Fragments

Until now, we described the classes that provide the framework for the application however the application still consists of blank pages. In this section we discuss how those empty pages are going to be filled with content.

The Zeeguu application needs to manage the different fragments in a intuitive way so that the user can easily switch between them.

Google suggests a solution for this and provides the `SlidingTabLayout`<sup>10</sup> which displays a horizontal list of fragments at the top of the device through which the user can easily change between these fragments when he either clicks on a tab in the sliding menu or swipes to the left or right on the screen. Those features increase the application's intuitive handling, because Android users will recognize this pattern from other applications like the Google Playstore (see Figure 4.2) and therefore know how to use it. This is why it was implemented in the Zeeguu application (see Figure 4.1).

This sliding menu is created through the `SlidingTabFragment` which is created in the main activity and it is responsible for the view of the entire app except for the action bar and the `SettingsFragment` as they are directly managed through the activity. The `SlidingTabFragment` in detail creates and manages the `SlidingTabLayout` which is responsible for the tabs of the menu directly beneath the title bar and the `ViewPager` which shows the page content that is beneath the `SlidingTabLayout` (see Figure 4.3).

The `SlidingTabFragment` can either cache all fragments or create them anew everytime the user switches between the fragments. If it caches the fragments, the application will be able to switch fast between the single fragments but it needs more memory. On the other hand if it loads every fragment anew when they are changed, the application does not need much memory but it is slower.

Zeeguu caches all fragments because it guarantees speed and that is rated for the appli-

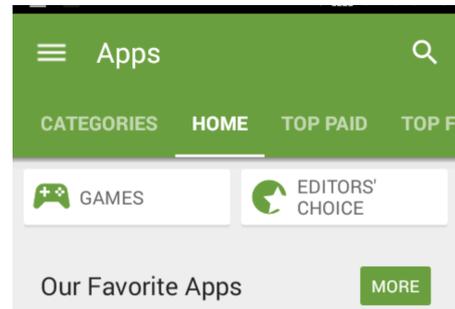


Figure 4.2: Google sliding menu in the Playstore

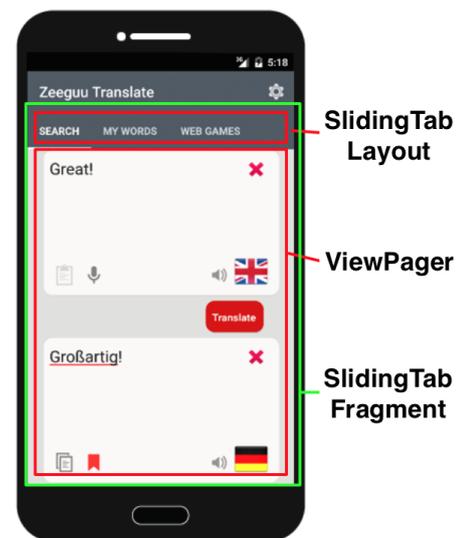


Figure 4.3: Sliding menu architecture

<sup>10</sup>Google, *Andorid Navigation Patterns*, <http://www.google.ch/design/spec/patterns/navigation.html>

cation more important than to save memory. This is how the user can translate a word, bookmark it, then instantly switch to the wordlist and see the word in there without any delay.

Like mentioned before Zeeguu restores or creates the fragments inside the Zeeguu-Activity. To keep the fragments inside the SlidingTabFragment and the Zeeguu-Activity in sync, the activity includes for every fragment which is implemented in the SlidingTabFragment a callback. Therefore when the SlidingTabFragment is created, it will create the sliding menu with the fragments that are collected through those callbacks to the ZeeguuActivity and thus the activity and the sliding menu classes will point to the same fragments.

The SlidingMenuFragment contains three fragments that will fill the view of the ViewPager: the SearchFragment, MyWordsFragment and ExerciseFragment. Here is an overview of those three fragments:

1. **SearchFragment:** it is responsible for the view that lets the user search for words or sentences. The main interaction with this fragment consists of three steps:
  - (a) The user can either enter text into the text field directly, paste his clipboard into it or speak into his microphone that will analyze what he said and paste it into the text field.
  - (b) To translate it into another language the user then needs to press the translation button.
  - (c) and if the user wants to expand his wordlist, he can bookmark the translation.

Furthermore the languages can be switched by a click into the other text field or changed by a click on the flags, a text field is read out loud when the loudspeaker button is pushed.

The main focus when this tab was designed was to clearly create recall value so that the user would instantly recognize what the tab is for and so what he can do with it. This focus influenced everything down to the last detail. The icons were chosen to be intuitive <sup>11</sup>. Text on the other hand was used only rarely, because symbols can be recognized faster (see Figure 4.4).

2. **MyWordsFragment:** defines the view that lets the user see his bookmarked words in an expandable list that is sorted by the date where the user searched for it. Little flags also indicate the languages of these translations.

The fragment includes many Android standards to improve the design towards an intuitive handling. Two examples are:

---

<sup>11</sup>The single icons were tested during the usability tests. It was evaluated if the users recognize the symbols and knew what action they would provoke. Symbols that were hard to interpret were changed during those tests

- The list itself can be actualized through a swipe towards the bottom at the top of the list. This is an established Android interaction pattern.
- Furthermore single translations can be deleted with a single click on the trash icon after entering the action view by a standard Android long click on an item.

The bookmarks are organized by day and in an ExpandableListView, which lets the user fold easily all the entries. In the future sorting options can be added to organize the list by other categories like word importance or article (see Figure 4.5).

3. **ExerciseFragment:** The last tab inside the sliding menu manages the view that allows the user to do exercises. Those exercises will ensure that the user will not forget his word collection and serve also to collect data through which the knowledge of the user can be evaluated.

The exercise tab is kept very clear and the instructions are easy visible (see Figure 4.6). The design of the exercises on the web was reused to encourage users that are already familiar with the web version of the application.

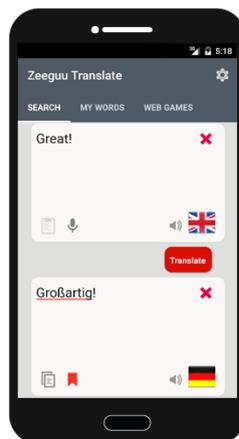


Figure 4.4: SearchFragment

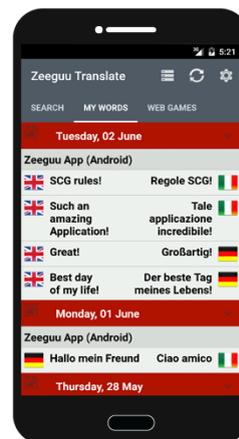


Figure 4.5: MyWordsFragment



Figure 4.6: ExerciseFragment

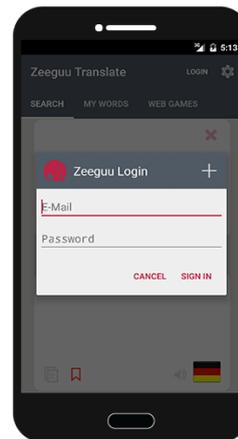


Figure 4.7: LoginDialog

### 4.3.1.3 Dialogs

Dialogs are used for login, create account, and logout processes.

- On login, a Dialog pops up where the user can enter his account information and after the confirmation button has been pressed, the Dialog will request a session id on the server through the help of a callback to the ConnectionManager. After the server responds, the user will either be logged in if the server sent a session id or the user will get an error message displayed with a new chance to login if the server sent an error message.
- The Dialog to create a new account works about the same, just that it asks next to the login information also for information about the new user and will then call a slightly different API but with the same answer. The response of the server is either the session id if the information is valid or an error message and a chance to try again if it was not. Both of those dialogs will check whether the entered information is valid or not and display error messages accordingly.
- The last Dialog is one to confirm the logout process when the logout button was pressed. It asks if the user really wants to logout or he just touched that button by mistake.

These Dialogs contain only the necessary information and this information is displayed in a familiar way to ensure that users know from other applications what needs to be done<sup>12</sup> (see Figure 4.7).

<sup>12</sup>The usability tests clearly stated that the users rated the dialogs as easy to understand.

## 4.3.2 Back-End

### 4.3.2.1 Using the Zeeguu Server API

The Zeeguu application needs to extend the Zeeguu ecosystem and therefore it has to communicate with the Zeeguu server. The Zeeguu server provides a REST API, which processes HTTP requests. The next section will give a brief overview about the server API and the functions which are needed for this application.

**/add\_user:** Creates a new account on the server  
Parameters: Email and password for the new account  
Response: Session id if user informations are correct

**/session:** Logs into an existing account on the server  
Parameters: Email and password of the existing account  
Response: Session id if login correct, otherwise an error

Every request from here on needs the session id. The session id is a unique number that is distributed by the server if you login with correct user information. Afterwards, the number serves as a clear verification for the user. The session id is always added at the end of the request URL and therefore will not be mentioned anymore.

**/translate:** Requests a translation for a word or sentences from one into another language  
Parameters: Word or sentence, context, URL and from/to language  
Response: JSONArray of the translated words with all possible meanings

**/bookmark\_with\_context:** Bookmarks a word or sentences and its translation  
Parameters: Word or sentence, title, context, URL and from/to language  
Response: Long value that represents the id of the word on the server

**/learned\_and\_native\_language:** Requests the languages of the user on the server  
Parameters: None  
Response: JSONArray of native and learning language

**/native\_language/learned\_language:** Changes the language of the user on the server  
Parameters: Which language to change and the language to set it to  
Response: HTTP 200 OK code

**/bookmarks\_by\_day:** Requests a list of your bookmarked words  
Parameters: None  
Response: JSONArray with all words ordered by day

**/delete\_bookmark:** Removes a bookmarked word  
Parameters: The id of the bookmark  
Response: HTTP 200 ok code

### 4.3.2.2 The Interface to the Zeeguu API

The `ConnectionManager` belongs to the back-end and handles all communication with the server through the server API that we have seen before. It uses the Volley Library provided by Google to submit all requests to the server. Volley helps to write clean RESTful HTTP requests that are executed asynchronously on a different thread, because from Android version HoneyComb onwards all HTTP requests need to be asynchronously or the application throws a `NetworkOnMainThreadException`.

The `ConnectionManager` also casts the `ZeeguuActivity` into an interface through which it can communicate with the main activity through a callback as you can see in Figure 4.8. It uses these callbacks to inform the different fragments when certain requests have been answered.

The `ConnectionManager` furthermore has access to the `Account` class, which handles all the data belonging to the user, starting with his e-mail and ending with his words.

As soon as the `ConnectionManager` is created, it creates an `Account` object and this `Account` object then loads all data that is saved locally except for the user words. If a session id is already active from the last time the application was used, it will re-enter it, if not, it reconnects to the server and asks for a new session id. For every communication with the server except the login and the create a new user request, a session id is required.

The `ConnectionManager` will additionally load the list of the user words after it has been created. It will therefore create a request after the session id has been acquired and send it to the server. If the server responds, the words will be formatted and cached in the `Account` object, which also will save all the words locally on the phone.

But if out of any reason the connection to the World Wide Web cannot be established, the `ConnectionManager` will call the `Account` object and tell it to load the words locally that have been acquired through the last synchronization.

As soon as the list is in the correct form, the `ConnectionManager` will inform the `ZeeguuActivity` through a callback to the interface that the list has changed and the `ZeeguuActivity` then will inform the `MyWordsFragment` that it has to actualize the view.



# 5

## The Validation

It is no easy task to validate that a software program solves the problem discussed in the beginning. The way to do it is to analyze it through two different perspectives: The first perspective will compare the goals of the beginning of this thesis with the actual achieved work and analyze which goals have been fulfilled and which not. The second perspective will evaluate how users rate the design and usability of the application.

### 5.1 Zeeguu Application Goal

The goal of this bachelor thesis was to create an application that allows the user to use the Zeeguu ecosystem everywhere and at anytime in order that he can profit from the SSR advantages at any time.

The application is currently in the beta test phase in the Google Playstore and can be downloaded by people who are granted access. The application fulfills all of the goals that were set in the beginning of the thesis with one small exception: The OCR functionality. It supports Android versions 4.0 (SDK: 18, ICE\_CREAM\_SANDWICH) and higher. The application was tested on multiple smartphones (Samsung Galaxy S3, Samsung Galaxy Note 2 and the Google Nexus 5) and on the emulator on different Android versions ranging from Android 4.0 to 5.1 (SDK: 22, LOLLIPOP\_MR1) and works as expected. The latest source code can be found on Github [7].

That means that the Zeeguu platform was successfully extended by an application for the Android phone that meets the requirements and even provides some more features that were added during the development phase. One example is the feature that the application

even provides functionality if the mobile phone is not connected to the internet. This feature includes that the wordlist is cached on the phone to load it later if the user is offline or a search feature that searches in the personal word library if the phone is not connected to the internet.

## 5.2 Usability tests

Usability tests analyze mainly the user interface and the interaction of the user with it. They are more or less a simulation of situations that users would encounter daily when using the application.

### 5.2.1 Methodology

Usability tests will take place between a potential user and the usability tester. The tester will place the framework for the test and afterwards give goals for the user to achieve. Important is that not a step-by-step manual is provided but a bigger target that the user wants to achieve. The result is to see if the user figures out the steps he has to take to reach the goal.

We furthermore told the user that he cannot make any mistake because the application is tested and not the user and that he is encouraged to say out loud what he is thinking because this will provide further information about the perception of the application. Through these tests a pretty good image of the design can be obtained.

In order to achieve the best result for the Zeeguu application the development process was accompanied by multiple usability tests which took place with some weeks space between each other. Through this measurement, there was enough time to evolve the application until the next usability test to see if the new version is an improvement in regards of intuitive handling and appealing design compared to the old one. Of course in most of the cases it was, because the information of the previous test was used to improve it.

This work included 4 usability tests (see Appendix C) where users were instructed to either use a test account or to create a new account to fulfill the usability goals. After being logged into Zeeguu, the users were given different goals combined with a small story for some background information. For example in one story was the user reading a newspaper article and he just realized that there is a word that he does not understand, or the user just encountered a difficult word for the third time and still does not know it. Afterwards the reaction of the user is analyzed.

### 5.2.2 Results

The first results were mostly about major design questions like how to change languages which feature icons to use and where to put them or how to switch between the fragments. But the more iterations took place, the less the feedback went into the direction of design patterns and the more it took off in directions of logic incongruences, bug detection and new possible features.

Furthermore users mentioned more and more that they liked some common design patterns (for example to change the languages click on the language flag) and it took them often less time to achieve the goal thanks to those patterns. The strongest effect was noticeable when a design pattern belonged to the Android standard patterns and so the users already knew how to use it.

This is a sign that the Zeeguu user interface improved constantly from iteration to iteration and was finally mostly rated intuitive and appealing.

Hence at the end of this chapter we can state that the goals of the application were achieved and the usability tests successful passed which means that the Zeeguu application can be validated.

# 6

## Conclusion and Future Work

This chapter will have a critical look at the entire work, reflect the experiences that were gained and in the end will open the floor for future work.

### 6.1 Conclusion

The Zeeguu application provides an easy way to use Zeeguu on an Android device through which it increases the likelihood of success of the entire Zeeguu platform. But nevertheless during the development phase many concepts and ideas changed and also the limits between what is deliverable and what is not were discovered more than once.

In the beginning of the thesis multiple applications for Android were planned which would support each other to give the user many different instruments which he can use to enjoy the Zeeguu experience. But as time and people showed during the usability tests, users do not want different applications from the same platform on their device, they only want one that allows them to do all they want in the simplest possible way.

The Zeeguu platform supports many different scenarios and it is difficult to provide all those features consolidated in one application. Thus there were multiple discussions whether to merge all into one application or keep multiple small ones. At this moment is still not clear if for the user's sake there will be finally only one single program they could install on their phones or if still multiple apps will be provided.

A further conclusion is that iterative development is an appropriate paradigm to program a mobile application if the platform itself is still in the development phase. For example the feedback that will be delivered to the user is not yet implemented in the

application because it is just not available on the server yet. And this feature is a very important one that will be a future strength of Zeeguu. This means that the development limits and the speed for such a project are defined through the status of the related platforms.

Another critical point about the application is that it allows the user to translate difficult words when reading a book or the newspaper and bookmark them, but it does not allow one to read texts on the web. This can lead to many situations where the user is not able to profit from the full aspect of the SSR advantages, especially in situations where he cannot access the browser plug-in which would slow down the learning progress in the end.

## **6.2 Future Work**

Zeeguu is a big project with ambitious goals. Therefore the platform provides lots of extending work that can be done.

### **6.2.1 Diversify into iOS and Windows Mobile**

The Zeeguu Android application covers most of the potential Zeeguu users who want to use the Zeeguu ecosystem on their smartphone because it is by far the most sold operating system for mobile devices in the whole world. Nevertheless there is still a considerable number of people who use Apple's iOS or Microsoft's Windows Mobile operating system and especially those people are hard to win if the mobile application cannot be provided. And especially in the beginning a platform needs to increase its chances of success by any means. Thus it would be critical not to serve the other two platforms because exactly those people who were excluded could make the difference between failure and success.

Therefore an application for those two platforms would be optimal and help Zeeguu to increase its value.

### **6.2.2 Improving the Zeeguu Android Application**

#### **6.2.2.1 Different Exercises**

Even though the Zeeguu Android application itself provides already many useful and smart tools, it still has huge improvement potential. The exercises provided until now only a simple translation workout, where the user sees the native word and a sentence in the learning language and he has to recognize the translated word in that sentence and type it into a text field. This serves the repetition and will bring some improvement, but it will be single sided in the long run and the users will probably stop doing those exercises

after a while. Thus different kind of exercises could be integrated with a daily changing rhythm to keep users interested and playing. Furthermore the training only takes place through one sensory organ, the eye. But hearing or speaking to increase the effectiveness of the training is never used and could be integrated to boost the recall rate of the user even further.

#### **6.2.2.2 Display Statistics and User Information**

At the moment the user can manage his words and do some exercises. In the near future, it is planned that he can even get statistics out of the Zeeguu platform, which will reflect his learning behavior, and this should motivate the user to keep learning and training. This could be integrated as well in the mobile version in form of a tab which would display the users account information along with his statistics.



# Anleitung zu Wissenschaftlichen Arbeiten

Zeeguu is an ecosystem that supports the user to learn new languages. The Key idea is that Zeeguu not only translates words or sentences into other languages, but also keeps a record of the translated and bookmarked words and creates exercises based on those words to increase the efficiency.

Recently the Android application extended the Zeeguu ecosystem so that the user can use it wherever and whenever he wants to. But since the Zeeguu platform is a very young offspring, there are many development processes in progress which lead to new features that will be implemented into the platform sooner or later. Often these new features need to be implemented in the mobile version as well.

Therefore this script provides guidance in form of a manual for programmers who want to extend the Zeeguu application. It contains step-by-step guides to extend single features with the goal to facilitate the further development process.

*“To improve is to change; to be perfect is to change often”. - Winston Churchill*

## **A.1 Support new Languages**

The Zeeguu ecosystem has still a long journey full of improvements ahead of it. And as a matter of fact it is a language learning platform which means that it is inevitable in the future to expand the supported languages.

Thus if the ecosystem updates the languages, the application should update them too. Therefore this chapter will provide a closer look at how this can be achieved.

### A.1.1 Modify the Language Arrays

The first adjustment that we have to modify is the strings.xml file. We have to add the new languages as items in the two language arrays. One of these arrays is responsible for the name of the languages, which are displayed to the users, while the other array holds the language keys, which are used by the server and the application logic. The language key values are ISO 639-1 standard language shortcuts and can be looked up for example on wikipedia [8]. Furthermore it is important that the item order of the two arrays is the same in both, so if the German language is the third item in one array, it should be also the third item in the other array.

If you would start the application right now and open the dialog to change the languages, you would already discover your newly implemented languages and the translations would probably work if it is already implemented on the server, but those languages are not represented by any flag yet. And so we have our next goal, to enter a flag.

```
<!-- languages arrays -->
<string-array name="languages">
  <item>Deutsch</item>
  <item>Dutch</item>
  <item>English</item>
  <item>French</item>
  <item>Italian</item>
  <item>Portuguese</item>
  <item>Spanish</item>
  <item>NewLanguage</item>
</string-array>

<string-array name="language_keys">
  <item>de</item>
  <item>nl</item>
  <item>en</item>
  <item>fr</item>
  <item>it</item>
  <item>pt</item>
  <item>es</item>
  <item>newLanguagekey</item>
</string-array>
```

Figure A.1: Modify language arrays

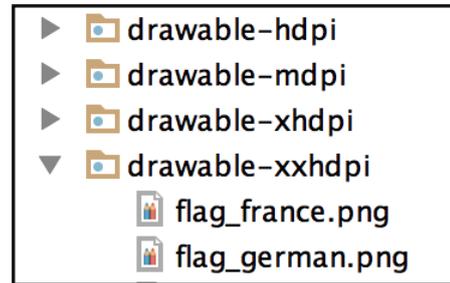
### A.1.2 Implement all Flags

One strength of the application is the intuitive design. One major reason that guarantees this intuitiveness is that a lot of symbols and icons were implemented in order that the user understands the user interface as fast as possible. This is the reason why flags are so important.

To let those flags appear in the program we need first to import the picture of those flags into the code. Android is designed to support all its devices and so it needs to be able to launch the application on phones that have just a few inches until tablets with over 10-inch displays. Therefore Android also supports different icon resolutions, so that on a tablet the icons still look very clear. But this also means that different icon resolutions need to be embedded in the code to support Android's scalability and therefore four drawables for one flag need to be implemented which differ in their resolution.

This means that the flags in the application are divided into different resolution folders:

- **drawable-mdpi:** 32 x 32 pixel
- **drawable-hdpi:** 48 x 48 pixel
- **drawable-xhdpi:** 64 x 64 pixel
- **drawable-xxhdpi:** 96 x 96 pixel



Thus for every flag implemented, the four drawables have to be put in the correct folder in the zeeguulibrary. The recommended and actual format of those flags is the portable network graphics or PNG and should be used on top of that. Important as well is that the flags inside the pictures almost touch the borders of the picture, because otherwise they are smaller than the flags which are already implemented and that looks awkward.

Figure A.2: Drawable folders in the Zeeguu project

Now the flags are in the code, but the application will not display them yet, because it does not know where to put them. We still need to refer the flags to the languages they represent in order that the right flag is shown in front of the language.

To do so we need to open the MyWordsItem class in the zeeguulibrary. There the static setFlag method can be found that clearly states which flag belongs to which language key. This class needs to be extended.

For every flag added, it needs a new “case” statement in the switch function. The identification value for this case statement must be the language key value which was entered in the strings.xml file.

The new body of our statement then only needs to set the image source of the ImageView that we get from the method to the flag that it belongs to.

And this is the end of the story. The new language has now successfully been added and the flag will be shown at all the right places.

```
public static void setFlag(ImageView flag, String language) {
    switch (language) {
        case "en":
            flag.setImageResource(R.drawable.flag_uk);
            break;
        case "de":
            flag.setImageResource(R.drawable.flag_german);
            break;
        case "fr":
            flag.setImageResource(R.drawable.flag_france);
            break;
        case "it":
            flag.setImageResource(R.drawable.flag_italy);
            break;
        case "nl":
            flag.setImageResource(R.drawable.flag_netherlands);
            break;
        case "pt":
            flag.setImageResource(R.drawable.flag_portugal);
            break;
        case "es":
            flag.setImageResource(R.drawable.flag_spain);
            break;
        case "newLanguagekey":
            flag.setImageResource(R.drawable.my_flag);
    }
}
```

Figure A.3: This method references a flag to every language key

## A.2 Manage the Sliding Menu

Another major extension node is the sliding menu that can be expanded by adding a new fragment. For example a RSS reader fragment could be integrated that shows all RSS feeds nicely in a list. But to keep the application as intuitive as possible, the sliding menu should not get too many entries. It is not clearly arranged if you have to swipe multiple times from the right to the left through the menu just to reach your desired fragment.

Thus if the application grew further in respect of the number of fragments, you would need to change the design that manages the fragments from a sliding menu to a navigation drawer that can be slided in from one side of the phone shortly afterwards. Google recommends to change to a navigation drawer if the application implements more than four fragments into the sliding menu [9].

To see how the `SlidingTabFragment` could be reworked or changed, let's have a look how we would add and remove a fragment. If you are able to do this, you will also be able to replace fragments as this is just a combination of removing and adding a fragment.

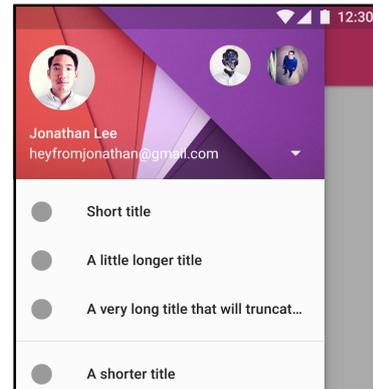


Figure A.4: Google's example of a navigation drawer [10]

### A.2.1 Create a new Fragment

The first step to add a new fragment is to create a new `Fragment` class. This is a pretty simple process as it is the same as in other applications. First you have to create a `NewFragment` class that extends the `Fragment`, afterwards you need to create the layout you want it to look like and implement this layout finally in the `onCreateView` method of this `NewFragment` class. Now you are already able to extend the features of your fragment and implement it afterwards into the sliding menu.

### A.2.2 Add a Fragment

To add a fragment, we first need to define it in the `ZeeguuActivity` as this is the place where all fragments will be created, also the ones from the sliding menu. First create two class variables. The first one is able to reference the fragment itself while the second one is a static integer that represents the id of the new fragment. It can be any value except those that are already used.

Afterwards we need to create the new fragment object when we start the activity in the `onCreate` method (see Figure A.5). To do so we first need to check whether it is already in the `FragmentManager` or not. To identify if the fragment is in the `FragmentManager`, we use the id variable that we created for the new fragment. If it is, the application was

already active before and we can use the old fragment in order that the user will not lose the information he already entered. If it is not, then the application was started from scratch and we create a new fragment.

```
newFragment = (NewFragment) fragmentManager.findFragmentByTag(getFragmentTag(ITEMIDFRAGMENT));
if (newFragment == null) newFragment = new NewFragment();
```

Figure A.5: Load an instance of the fragment

Now we create the fragment when we start the application and load it every time we reload the main activity. But it is not yet in the sliding menu. To get the fragment into the sliding menu, we first need to add a method in the SlidingFragmentCallback interface A.6, which is located in the SlidingTabFragment class, that allows us to get the new fragment through a callback.

After the modification of the interface we need to implement this new method in the ZeeguuActivity and let it return the NewFragment that we added before (see Figure A.7).

Now we finished our work in the ZeeguuActivity and need to extend the SlidingTabFragment class next. In this class you will find a list called tabs. Every tab entry in this list is in the end one page in the sliding menu. And therefore we need to add our fragment to this list, which need to be done in the onActivityCreated method.

The list includes PagerFragmentTab items and this means we need to create a new PagerFragmentTab object that represents our fragment. The constructor of this class needs the following information:

```
PagerFragmentTab(long itemId, CharSequence title, int indicatorColor,
Fragment fragment)
```

The itemId is the same id as we defined in the ZeeguuActivity and we can reference it easily if it is static by just calling ZeeguuActivity.NAMEOFTHEFRAGMENTID.

The title attribute is nothing else than the title of the tab in the sliding menu and the indicatorColor integer is the color code that the tab should adopt. Since all the tabs have the same color, this attribute can just be copied from another fragment that is already added. The last attribute is the fragment that we want to add and it is received when we execute the callback.getNewFragment method.

```
public interface SlidingFragmentCallback {
    SearchFragment getSearchFragment();
    MyWordsFragment getMyWordsFragment();
    ExerciseFragment getExerciseFragment();
    BrowserFragment getBrowserFragment();
    NewFragment getNewFragment();
}
```

Figure A.6: Extend the SlidingFragmentCallback interface

```
@Override
public NewFragment getNewFragment() {
    return newFragment;
}
```

Figure A.7: Implement the method in the ZeeguuActivity

```
@Override
public void onActivityCreated(Bundle savedInstanceState) {
    super.onActivityCreated(savedInstanceState);
    callback = (SlidingFragmentCallback) getActivity();

    tabs.add(new PagerFragmentTab(
        ZeeguuActivity.ITEMIDFRAGMENT,
        "NewFragmentTitle",
        getResources().getColor(R.color.sliding_menu_line),
        callback.getNewFragment()));
}
```

Figure A.8: Add a tab to sliding menu adapter

We get about the same code as in Figure A.8 that allows us to insert the new fragment. Keep in mind that the order of the items in the list represents also the order in the sliding menu. If you add the item in the end to the list when all other items already have been added, it will also appear as the last entry in the sliding menu.

One last thing may be changed at this point. The numbers of cached fragments that the sliding menu keeps is defined in the `SlidingTabFragment` class after the tabs are filled. The method looks like this:

```
viewPager.setOffscreenPageLimit(3);
```

Here you can change the integer to match the number of screens you want to have cached parallel in the smartphone while switching between the tabs. Of course the more you cache, the faster the application will be, but the more memory it needs and vice versa.

And voilà, a new fragment has been added to the sliding menu and if you start the application, it will look similar to Figure A.9.

### A.2.3 Remove a Fragment

Removing a fragment from the sliding menu is basically the same as adding a fragment except that it is done in the inverse order. First delete the code that will add the fragment to the tab list in the `SlidingTabFragment` class. After this step your fragment will still be loaded and therefore needs capacity, but it does not appear anymore in the sliding menu.

To remove it completely from the application, delete the class variables of this fragment in the `ZeeguuActivity` and afterwards in the `onCreate` method the code where you create the object. In the end you just have to delete the getter method for the fragment in the `SlidingFragmentCallback` interface and then also delete the get method for the fragment you implemented in the `ZeeguuActivity`.

If you want you can also delete the classes which belong to the fragment inside the project to finally get rid of all the code of that fragment. The last thing that can be done is to correct the number of all cached fragments in the sliding menu. Please look in the section above how to do so.

By now your sliding menu has an entry less and all the traces of it have been deleted.

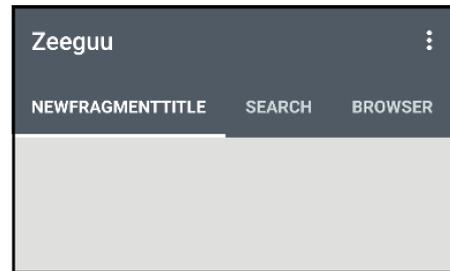


Figure A.9: The sliding menu with the empty fragment

## A.3 Create a new Dialog

To let a new dialog pop up, some work has to be done. The first step is to create a new Dialog class, which extends itself from the DialogFragment.

It consists of at least an onCreateDialog method as seen in Figure A.10, which defines the body of the Dialog and the action that it has to take when certain buttons are pressed. If needed, other methods can be added or overwritten to extend the dialog even more.

For example with an onAttach method, you can implement an interface to communicate with the Activity as we see in the next chapter.

After you have defined the dialog, you can create a method in the main activity that lets you call it. This method needs to create a new object of these dialogs, then set its parameters if needed and in the end call the show method upon it. It looks similar as in Figure A.11.

You are now able to let the dialog pop up by just calling the single method in the Zeeguu-Activity. Also fragments can call the method with the help of interfaces as we will see in the next chapter.

```
public class ZeeguuLogoutDialog extends DialogFragment {
    private ZeeguuDialogCallbacks callback;

    @Override
    public Dialog onCreateDialog(Bundle savedInstanceState) {
        AlertDialog.Builder builder;
        if (android.os.Build.VERSION.SDK_INT >= Build.VERSION_CODES.LOLLIPOP)
            builder = new AlertDialog.Builder(getActivity(), R.style.AlertDialogCustom);
        else
            builder = new AlertDialog.Builder(getActivity());
        builder.setTitle("Do you really want to log out?");
        builder.setCancelable(true);
        builder.setPositiveButton("Zeeguu Logout", (dialog, which) -> {
            callback.getConnectionManager().getAccount().logout();
        });
        builder.setNegativeButton("Cancel", (dlg, which) -> {
            dlg.cancel();
        });
        return builder.create();
    }
}
```

Figure A.10: Example of a Dialog class including the onCreateDialog method

```
@Override
public void showZeeguuLoginDialog(String message, String email) {
    ZeeguuLoginDialog zeeguuLoginDialog = new ZeeguuLoginDialog();
    zeeguuLoginDialog.setEmail(email);
    zeeguuLoginDialog.setMessage(message);
    zeeguuLoginDialog.show(fragmentManager, "zeeguuLoginDialog");
}
```

Figure A.11: Example of a method in the ZeeguuActivity that opens a Dialog

## A.4 Fragment Communication

The communication between the fragments happens through interfaces. Every fragment creates its own interface with all the methods it needs and this interface is afterwards implemented in the ZeeguuActivity. Then the ZeeguuActivity implements all methods from the interface.

The fragment is now able to cast the activity into his interface and call those methods. Therefore create a class variable that references the callback variable, which you have casted into the interface, so that you will be able to access it anytime. Thus in the onAttach method of the fragment, you can cast the activity that you get into this interface which will look something like:

```
callback = (ZeeguuNewFragmentInterface) activity;
```

The activity variable inside the code above will be delivered by the method and the callback is the name of the class interface variable. Now you will get as soon as

the fragment is attached to the activity a working connection to communicate with the activity.

If now our `NewFragment` for example wanted to message fragment B a string, it would execute the `callback.sendFragmentB("string")` method which would be implemented in the fragments interface and so the method would also be implemented in the `ZeeguuActivity`. Thus the `sendFragmentB(String string)` method in the main activity would be called and this method then on the other hand would call a method in that fragment B that would take the string and process it.

The reason why the main activity can refer to the single fragments pretty easy is because it manages already all fragments.

## A.5 Add new API requests

New API requests can be defined in the `ZeeguuConnectionManager` as this class is responsible for all connections to the server. This class uses the Android's Volley library [11] to create and manage requests. This makes it pretty easy to send REST request to the server and afterwards process the answer.

For every new request a new method should be created in the `ZeeguuConnectionManager` to keep the code as clean as possible. The method itself needs to know the URL to which it has to send the request and what request type it is [12].

Volley provides 3 request types:

1. **StringRequest**: is a basic request that receives a plain string.
2. **ImageRequest**: is a request that receives an image that it can access through the URL.
3. **JsonRequest**: is a request that receives a Json file. Two classes inherit from this class:
  - (a) The **JsonObjectRequest** that receives just a single `JsonObject` and
  - (b) The **JSONArrayRequest** which receives a complete `JSONArray`.

The decision which request type we take is dependent on the response type of the server. For example if the server sends back a plain string of information, a `StringRequest` needs to be used because this request can handle the answer of the server while the others cannot.

Furthermore each request type is divided into different request methods. GET and POST are the most common types and the ones that are used on the server. The GET method is used when only data is requested but not committed to the server. On the other hand the POST method is used to submit data to the server, which then is to be

processed. Other possible methods would be HEAD, PUT, DELETE, OPTIONS, PATCH and TRACE, but they will not be further explained in this paper as they are not used on the Zeeguu server yet.

Lets take the previous example and create a new `StringRequest`. Like already mentioned we want to create the method in the `ZeeguuConnectionManager` first. Therefore we create the public void `newRESTRequestToServer()` method, which should get some information of the server when it gets called.

In the beginning of the method we will make some assertions that check whether all preconditions are met, for example if all inputs are correct or if other similar actions are already running.

After that, we will create the URL for the request. The URL consists of the static address of the server that does not change, the dynamic part that changes from request to request and some parameters. Our request URL will look something like this:

```
String urlNewRESTRequest = URL + "pathOfRequest" + "?session="
+ account.getSessionID();
```

The new variable `urlNewRESTRequest` will save the complete URL which consists of:

1. The static address of the server that will be the same for all requests. In our case it will be the address of the Zeeguu server at the University of Bern and therefore the first part of the address is “`https://www.zeeguu.unibe.ch/`”.
2. The dynamic part of the address says which API you want to access on the server. In our case we want to connect to the `pathOfRequest` method.
3. In the end we want to add some parameters to identify ourselves. Here the session id will be added. To guarantee that the server will recognize this variable, we also need to put the name of the parameter in front of the session id. This is done with the “`?session=`” tag followed then by the session id.

Now that we have saved the URL, we need to create the request itself. We want to receive a string and so we will create a `StringRequest`. As parameters we need to pass the following parameters:

```
StringRequest request = new StringRequest(Request Method, URL,
Response Listener, Response Error Listener);
```

Let us analyze the single variables a little bit closer:

1. **Request Method:** This parameter defines, as we have heard before, what the request intends to do on the server. In our case it is only a GET method, because we will request data from the server but we do not commit anything.
2. **URL:** This is the URL that we have determined before and we can just pass it as a string to the request.
3. **Response Listener:** This listener defines what to do when the answer from the server arrives.
4. **Response Error Listener:** The error listener defines what to do if an error occurs during the process and how this error has to be handled.

After we have created the request with all those parameters, the only thing that is left to do is to tell the Volley library that the request is ready to be sent. To do so, we need to add the request to the RequestQueue and we are done. All the rest will be handled through the two listeners we defined a few lines ago. They will handle the answer when it arrives or catch the error if it emerges. And finally the complete method will look similar to Figure A.12.

If we would like to create a POST request, we furthermore may need to add some POST variables. To do so, we can overwrite a method of the request itself called:

```
protected Map<String, String> getParams();
```

This method returns a Map of all POST variables. Just add all the variables that you want to send with their keys like in Figure A.13 to this Map and return it. Now all those parameters will be added to the body of the request.

```
public void newRESTRequestToServer() {
    if(assertions())
        return;
    String urlNewRESTRequest = URL + "/pathOfRequest" + "?session=" + account.getSessionID();
    StringRequest request = new StringRequest(Request.Method.GET,
        urlNewRESTRequest, new Response.Listener<String>() {
        @Override
        public void onResponse(String response) {
            //Put in your code here when the request was successful
        }
    }, new Response.ErrorListener() {
        @Override
        public void onErrorResponse(VolleyError error) {
            //Put in your code here when the request was unsuccessful
        }
    });
    queue.add(request);
}
```

Figure A.12: Example of a new API request in the ZeeguuConnectionManager

```
@Override
protected Map<String, String> getParams() {
    Map<String, String> params = new HashMap<>();
    params.put("password", password);
    return params;
}
```

Figure A.13: Example of POST variables

# Bibliography

- [1] Kahneman, Daniel. *Thinking, Fast and Slow*. United States: Farrar, Straus and Giroux, 2011.
- [2] Marti, Simon. *ZeeGuu - A Platform for Second Language Acquisition Through Free Reading and Repetition*. Bern: University of Bern, 2013.
- [3] Krashen, Stephen. “False Claims About Phonemic Awareness, Phonics, Skills vs. Whole Language, and Recreational Reading”. <http://www.nochildleft.com/2003/may03reading.html>, 2003.
- [4] Shankland, Stephen. CNET. “Google Translate now serves 200 million people daily”. <http://www.cnet.com/news/google-translate-now-serves-200-million-people-daily>, 2013.
- [5] Publishing Technology. “Reading on mobile phones may account for nearly 5% of all book consumption in the UK”. <http://www.publishingtechnology.com/2014/10/reading-on-mobile-phones-may-account-for-nearly-5-of-all-book-consumption-in-the-uk>, 2014.
- [6] IDC. “Smartphone OS Market Share”. <http://www.idc.com/prodserv/smartphone-os-market-share.jsp>, 2015.
- [7] GitHub. “Zeeguu Android Source Code”. <https://github.com/SimpleDay/ZeeGuu.git>, 2015.
- [8] Wikipedia. “List of ISO 639-1 codes”. [https://en.wikipedia.org/wiki/List\\_of\\_ISO\\_639-1\\_codes](https://en.wikipedia.org/wiki/List_of_ISO_639-1_codes), 2015.
- [9] Google. “Andorid Navigation Patterns”. <http://www.google.ch/design/spec/patterns/navigation.html>, 2015.
- [10] Google. “Navigation Drawer Patterns”. <http://www.google.ch/design/spec/patterns/navigation-drawer.html#navigation-drawer-content>, 2015.

- [11] Android Developer. “Transmitting Network Data Using Volley”. <https://developer.android.com/training/volley>, 2015.
- [12] Android Developer. “Making a Standard Request”. <https://developer.android.com/training/volley/request.html>, 2015.

# B

## Designing User Interfaces on Android

There are many Android specific terms, which are important to understand the structure and realization of the application. Therefore this chapter will give a brief overview about the most important keywords.

### B.1 Activity

The activity class is one of the most important classes when programming with Android. Every application needs to be based on at least one activity. Thus if a user clicks on a specific application in his homescreen, the operating system will check which activity is defined as the main one in that application and then will start it.

Activities are responsible for the user interface, which means basically the content that is shown on the screen, and the logic, which is all the work that happens in the background. Those two responsibilities can either be taken by itself or the activity can make a fragment responsible, which will then handle it.



Figure B.1: Application with two activities

## B.2 Fragment

Fragments got implemented in the Android operating system during the update to the API level 11, which was also called Android 3.0 or Honeycomb. Fragments can be thought of as “small activities” because they need way less memory and calculation time when they get created instead of an activity. Fragments also provide a user interface and are able to do some background logic calculations like a regular activity can. The only disadvantage is that a fragment needs an activity in the background so that it will work, because fragments of course lack many functions to keep them slim and fast and those functions are substituted through an activity that it belongs to. The advantage is that for an application which has more than one screen, not multiple activities are needed anymore, but one activity with multiple fragment (see Figure B.2).

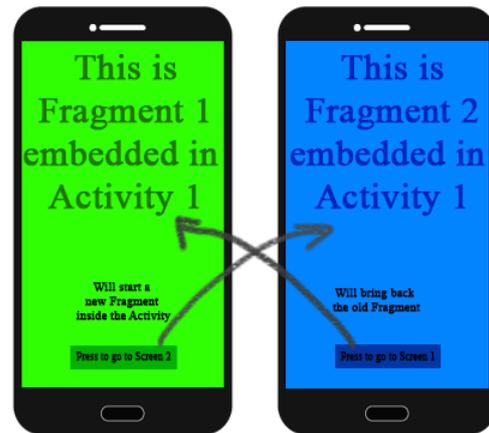


Figure B.2: Application with one activity and two fragments

Most application that use fragments use activities just to manage its fragments and the communication between them. For everything else, they have the fragments that do the work. This makes an application very dynamic and also very fast at the same time.

## B.3 Dialog

Dialogs are popup windows that cover the actual content and so get the focus of the user. They can be used either to display information, request information or as a confirmation tool.

## B.4 Interfaces

Interfaces bind classes together and define common functions that every class needs to provide if it implements the interface. In Android this is often used to communicate between the fragments. Therefore the activity that handles all fragments implements an interface that provides some methods which on the other hand inform the targeted fragments. Then fragment A for example can cast the activity into the interface, which means that it then is able to call the methods of the interface through a callback to the interface which will then tell fragment B that something happened.

The advantage of using interfaces is that it will make the single fragments independent of each other. This happens because fragments do not have to call each other directly anymore, instead they use callbacks to call the activity and the activity will handle the rest for them. Therefore fragment A does not reference fragment b directly anymore, but will call the activity and tell that it wants to do this action and then the activity will afterwards reference the other fragments.

Next to the clean code you get it also makes single fragments easily exportable into other projects, because you do not need to change the code just since you have project specific other fragments inside the code of the one fragment you want to export.

And furthermore if you want to import the fragment into an other project you can just add it in the code, then implement the interface that belongs to it in the activity, implement the methods and that is it.



# Usability Testing

## C.1 Initial Situation

Some users will be asked to use an account that already exists while other users will be asked to create a new account.

**Existing Account:**

E-Mail: tester@UsabilityTest.ch

Password: UsabilityTest

The account has a few words in the wordlist, but not the word “friendly”. Furthermore German or English is not chosen. The user will use his device if he got an Android device or use the test device if he has none. The application is reinstalled on the device so that the usability tester can open it for the first time.

## C.2 Test Results

### Haidar Osman (29.04.15)

Actions for the user to do:

- 1. Login to Zeeguu Account**  
User has to write email and password again when the login information is wrong, maybe just ask for password again.
- 2. Choose languages (English - German)**  
The user managed to switch and change the languages pretty fast. It seems to be intuitive.
- 3. Translate the English word “friendly” to German**  
Pretty fast and easy.
- 4. Copy the German translation, switch language, paste the German translation and translate it into English**  
The user did not get the paste symbol, whereas the copy icon was easy to understand.
- 5. Bookmark this word**  
Bookmark symbol should be in the middle of both fields because it bookmarks the combination.
- 6. Look up the bookmarked word in the wordlist**  
Swiping between Fragments is not intuitive for iPhone users. But nevertheless the action was pretty fast adopted.
- 7. Delete the word you bookmarked**  
The user first tried to swipe it away, but then found the option through a long click.
- 8. Other input**  
The font is too small to read it easily.  
Remove the user session ID from the settings.  
Settings that are not clickable should be set on disabled so that they are clearly distinguished.

**Thomas Steinmann (29.04.15)**

Osman Haidar demonstrated that if we tell the instructions, which the user has to fulfill, in form of stories it will push usability tests so that more information can be extracted instead of user manuals where every single step is listed. Out of this reason we changed the usability tests slightly. The user also gets a quick introduction into the Zeeguu ecosystem (not the application) to understand it.

1. **The user just downloaded the application because he encountered the word “friendly” in the newspaper and does not know what it means. Thus he wants to know the spanish word of it**  
Login process was clear for the user. He also understood how to change the languages, even though he first switched them. Therefore he suggested a switch button to switch the languages. Furthermore he was confused that the text field on the bottom could not be clicked on. But in the end he successfully translated the word in a reasonable time.
2. **The user wants to use the word in another application**  
The copy icon was found pretty quickly.
3. **The user wants to insert a word that he has in the clipboard from another application**  
The paste function from the Android IOS was used.
4. **The user did not know the word for multiple times and now wants to learn it definitely**  
The user found it pretty fast, but he asked why is it another color than the rest of the icons.
5. **The user wants to see if his word was bookmarked successfully**  
The user understood the concept of swiping between tabs or fragments.
6. **You do not want the word anymore in the wordlist**  
The user found the remove function pretty fast.
7. **Other input**  
The user did not like that the lower text field is hidden when the keyboard pops up. One further question was why the translation button did not shift when the keyboard opens.

**Julia Ebnetter (21.05.15)**

1. **The user just downloaded the application because he encountered the word “friendly” in the newspaper and does not know what it means. Thus he wants to know the spanish word of it**

The user did not see the button to create a new account. She did not also see the second language after she logged in because it was hidden by the keyboard. Furthermore she did not rate the switch language as intuitive. She finally managed to translate the word and proposed to that you could extend the translation and show for example if the word is masculine or feminine.

Bug found: switch languages changes languages into different ones.

2. **The user wants to use the word in another application**  
She understood it quite fast.
3. **The user wants to insert a word that he has in the clipboard from another application**  
She also used the paste function from Android IOS.
4. **The user did not know the word for multiple times and now wants to learn it definitely**  
She did not rate the bookmark sign as intuitive.
5. **The user wants to see if his word was bookmarked successfully**  
She immediately found the tab with her words and even used the “swiping” actualize function.
6. **You do not want the word anymore in the wordlist**  
She removed the word pretty quickly.
7. **Other input**  
The logout process is also very easy to understand.

**Sammer Puran (28.05.15)**

1. **The user just downloaded the application because he encountered the word “friendly” in the newspaper and does not know what it means. Thus he wants to know the spanish word of it**  
Installing the application on his phone failed the first time. After a few minutes he was able to run it. The button to create a new account is not intuitive, but the design of the dialogs is very appealing. The user got the impression that the keyboard hides a lot, maybe use a scrollview. The switch languages process was very easy and understandable for the user. There occurred a bug, one flag was not actualized when the language was changed.
2. **The user wants to use the word in another application**  
He understood it immediately.
3. **The user wants to insert a word that he has in the clipboard from another application**  
He used the paste icon of the application and found it pretty simple.
4. **The user did not know the word for multiple times and now wants to learn it definitely**  
Button was found, but he could push the word multiple times because of his slow connection.
5. **The user wants to see if his word was bookmarked successfully**  
He had displayed no words because of his slow connection, but also no buttons were displayed to actualize the wordlist.
6. **You do not want the word anymore in the wordlist**  
He found the remove function pretty fast but the application crashed when he pressed it. He also proposed a search for bookmarked words and to sort the wordlist with different criterias.
7. **Other input**  
The exercise tab does not tell you if a word was entered incorrectly. Also the setting button was not working sometimes and the application crashed when the user tried to logout. Also the logout dialog stated as a positive answer “Zeeguu logout” instead of a simple “Yes”.