FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

ASSIGNMENT OF MASTER'S THESIS

Title:	Grammatica - an app for practicing grammar
Student:	Bc. Alexander Bublik
Supervisor:	Ing. Filip Křikava, Ph.D.
Study Programme:	Informatics
Study Branch:	Web and Software Engineering
Department:	Department of Software Engineering
Validity:	Until the end of winter semester 2020/21

Instructions

Doing grammatical exercises is an essential part of learning any foreign language. However, working with the classical paper workbooks is rather tedious. The aim of this thesis is to develop an application, Grammatica, for Android or iOS tablets that will make working on grammatical exercise fun. The student will complete grammatical exercises by handwriting using tablet stylus having an immediate feedback from the app. The application shall support two types of exercises: partial/full sentence completion, filling a partial sentence and QA linking. It will automatically correct user input, keep track of his or her progress and allow it to safe in a cloud.

Analyze popular existing applications from either application store.
Design the application including the backend - data store and format for exercises and a way how compose together grammatical exercises.
Implement the app and the backend including sufficient testing and documentation.

References

Will be provided by the supervisor.

Ing. Michal Valenta, Ph.D.
Head of Department

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

Prague February 19, 2019

FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Master's thesis

Grammatica - an app for practicing grammar

Bc. Alexander Bublik

Department of Web Engineering
Supervisor: Ing. Filip Křikava, Ph.D

May 9, 2019

Declaration

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

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

In Prague on May 9, 2019 .....

Czech Technical University in Prague
Faculty of Information Technology

Citation of this thesis

Bublik, Alexander. *Grammatica - an app for practicing grammar.* Master's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2019.

Abstrakt

Tato práce popisuje proces vývoje mobilní aplikace pro operační systém Android a serverové části s databází, která komunikuje s klientem prostřednictvím REST API. Aplikace je určena pro studium cizích jazyku, podporuje rukopisný vstup a co nejpřesnéji simuluje zážitek ze cvičení v učebnici.

Práce obsahuje analýzu možných řešení a konkurenčních aplikací, návrh klientské a serverové části, implementaci a testování aplikace.

Klíčová slova mobilní aplikace, Android, rukopisný vstup, výuka jazyka

Abstract

This thesis describes the process of creating a mobile application for Android OS for studying foreign languages and server-side with a database that communicates with the client through the REST API. The application support handwriting input and simulates the experience of doing exercises in the textbook as accurately as possible.

The thesis paper contains an analysis of possible solutions and competing applications, designing client and server parts, implementation and testing of the application.

Keywords mobile application, Android, handwriting input, language learning

Contents

List of Figures

List of Tables

Introduction

Every year the world gets more and more connected, the ways of movement and communication between countries become easier and cheaper, more and more people begin to travel to other countries and communicate with people of another culture. Learning foreign languages is important by itself, but globalization is pushing people to learn them even more. Mobile learning applications are quite a popular segment of the mobile apps market. Moreover, in this segment, applications for learning languages show steady growth. [1] Moreover, given that there about 2.3 billion operating devices on Android [2], developing an application for learning languages looks like a good idea, since there is a demand for such applications.

While are many ways to learn foreign languages (e.g., tutors, language courses, language textbooks, communication with native speakers), learning languages using mobile devices has many benefits over traditional methods:

1. The phone is always at hand.
2. The lessons usually do not take much time.
3. The lessons are very interactive.
4. The addition of game elements allows to improve the efficiency of learning languages significantly.

Most mobile applications for learning languages use relatively new methods of learning, for example, the Leitner system [3]. However, there are people, who prefer a more traditional approach such as grammar exercises in textbooks. The purpose of this work is to develop an application that uses these more traditional teaching methods.

The main difference from competitors is handwriting - the goal was to simulate the experience of doing exercises in the textbook as accurately as possible. It will increase the visibility of the exercises and allow the person to

train the writing of a foreign language, which will increase the effectiveness of memorizing words and phrases.

In the first part of this work, we analyzed the existing similar applications, the existing handwriting input solutions, identified the functional requirements and the primary application use cases. The second part focuses on interface design and application architecture. Then, in the third part, the implementation is considered - the technologies and libraries used, the database structure and the server part. In the final fourth part, we define the testing methodology and run tests on real users with further application correction based on their feedback.

Chapter 1

Analysis

1.1 Platform Selection

Today, the market of mobile devices is a competition between the two main players: Android from Google and Apple's iOs. Each platform has its advantages and disadvantages. In this chapter, I will describe the strengths of each platform and the difference in the approach to development.

The Android operating system officially launched in 2008 and by April 2009, more than 1 million HTC Dream devices (T-Mobile G1), the first Android smartphone, were sold in the US alone [4]. However, the system widely spreads after 2010 with the release of the 2.3 Gingerbread update. Finally, Android "caught up" with iOs for the convenience and intuitiveness of the UI with the release of version 4.4 KitKat.

Since its release, Android has rapidly captured an ever-increasing percentage of the mobile OS market. Only recently this growth has slowed. To date (November 2018), the share of Android devices has reached 87.5% of the mobile OS market [5].

The next item that deserves attention is the vast variety of devices running on Android compared to iOs. Android can be selected as OS on devices of different manufacturers. Moreover, each of them can change and modify the system core and UI to fit its needs, because Android is an open source project. An Android device can have any screen size, resolution, and support old or new OS versions. Because many manufacturers modify the Android kernel, updating devices rest entirely on them, and usually, this is a very long process. Often, mobile devices are supported only 2-3 years, after which the updates cease to arrive. From here there can be problems with the support of non-standard devices and old versions of an operating system.

In iOs, on the contrary, the list of supported devices is quite small, system updates are reported in advance (usually in several months), and all devices are updated more or less simultaneously. It allows developers to tune applications for specific Apple devices.

Also of note is the simplicity of developing applications for Android and publishing them on Google Play. It is possible to develop Android applications on any platform - Windows, Linux, Mac and to publish on Google Play it is enough to pay a $25) fee once for a developer account registration [6]. To develop applications for iOs, a developer must have Apple devices (for example, iMac) and annually pay a subscription ($99 per year) to access the AppStore [7].

With all the above reasons, the Android platform is selected for application development and publishing.

1.2 Minimal API Selection

As mentioned in the previous chapter, the update of old Android devices to the new OS version is rather slow or not happening at all. Hence, the developer faces a severe problem of choosing which minimum Android version to support in his application.

At the moment (November 2018) for publishing in Google Play new apps and app updates must target at least Android 8.0 (API level 26), and developers should target API level 16 as the minimum supported level since Google Play will no longer update Play Services APK beyond version 15 for devices running ICS. Also, the oldest supported API for iInk handwriting recognizer is level 21. Therefore, I decided to limit the minimum supported API to level 21. As can be seen from Table 1.1, this will allow me to support almost 89% of all working Android devices [8].

Version	Codename	API	Distribution
2.3.3 - 2.3.7	Gingerbread	10	0.3%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	0.3%
4.1.x	Jelly Bean	16	1.2%
4.2.x		17	1.5%
4.3		18	0.5%
4.4	KitKat	19	6.9%
5.0	Lollipop	21	3.0%
5.1		22	11.5%
6.0	Marshmallow	23	16.9%
7.0	Nougat	24	11.4%
7.1		25	7.8%
8.0	Oreo	26	12.9%
8.1		27	15.4%
9	Pie	28	10.4%

Table 1.1: Android devices distribution

1.3 Existing Solutions

In this section, applications with a similar scope will be reviewed and analyzed. I will select some of the most popular language learning apps on Google Play.

1.3.1 Duolingo: Learn Languages Free

Author: Duolingo, Inc
Updated: April 23, 2019
Installs: 100+ millions
Price: Free with in-app purchases

Duolingo is one of the most popular services for learning foreign languages. According to the authors, Duolingo applies the most effective and modern teaching methods. I used Duolingo for about two years and made several observations. The training focuses on gamification - user can get badges, compete with other users and earn local currency - lingots, which can be spent for example on clothes for Duolingo mascot. However, as it seems, the essential element is streak count.

A fire icon at the top of the screen records how many days in a row you have studied a language on Duolingo.

All courses divided into levels; each level, in turn, is divided by topics.

Each lesson includes a variety of speaking, listening, translation, and multiple choice exercises.

However, Duolingo mainly aimed at beginners who want to learn the basics of the language. If the user already has good language skills and want to improve them, this application will not help. However, it is not necessary to start from the very beginning - inside the application, user can take the test, according to the results of which Duolingo will let him bypass a certain number of levels.

One of the downsides is that there are no explanations of grammar rules in Duolingo. It is assumed that the user will intuitively get these rules or look at them in some textbook. There are also many exercises that are too easy - a multiple-choice question is much easier than an open-ended question where the user has to come up with the answer himself. Moreover, there are no open-ended questions in Duolingo. Even when Duolingo asks the user to translate a sentence, it gives him a jumble of words to choose from, so he just has to put the words in the right order.

In general, if a student understands the basics of the language well and wants extra practice, this application can help. However, if a student is already at an advanced level, most likely he will be bored.

Figure 1.1: Duolingo user progress

1.3.2 Learn Languages with Memrise

Author: Memrise Limited
Updated: April 10, 2019
Installs: 10+ millions
Price: Free with in-app purchases

Initially, Memrise was just a flashcard program. It uses spaced-repetition software making it easy to learn new words and review them periodically. Also, although these cards still form the basis of the educational process, the company has also developed language courses that include other types of exercises. Some of those exercise types are only available with a premium account, but others are free to use. A distinctive feature of this application is that in addition to the "official" courses from the developer, users can create "community" courses. Anyone who signs up for a Memrise account can create

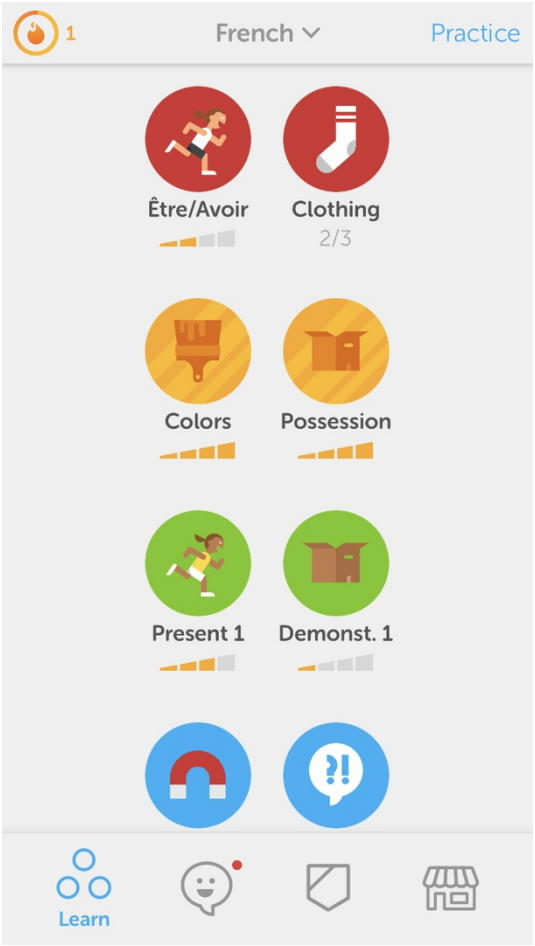

Figure 1.2: Duolingo levels

a community course, so the quality of these varies dramatically.

The application has some problems with the UI - recently in March 2019, a full rebranding was carried out - colors, logos, and UX were changed. Such a "restart" happens not for the first time in the history of this application. In recent reviews on Google Play, you can see a lot of dissatisfied people who do not like these changes.

So although now there are custom courses it is still basically a gamified flashcard app. Therefore, it should be used with other resources, not in isolation.

1.3.3 English with Lingualeo

Author: Google Commerce Ltd
Updated: March 13, 2019

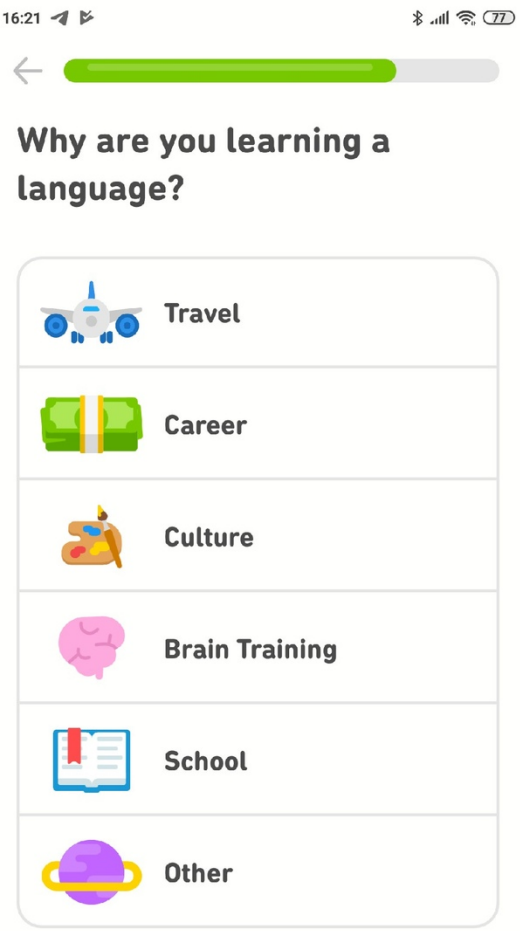

Figure 1.3: Duolingo exercise

Installs: 5+ millions
Price: Free with in-app purchases

Lingualeo is a convenient and free service for learning the English language. Over 18 million people worldwide use it.

The application contains exercises to expand vocabulary and game training to develop reading, writing and listening skills.

The training program is based on the level of knowledge of the language, interests, goals, and age of the student.

Modern teaching methods with strong gamification are used.

A student can see his or her progress in learning and adjust the development of certain skills.

The application is free, but to access the expanded functionality, the student has to purchase a premium subscription.

The emphasis in the application is made on learning the basics of the

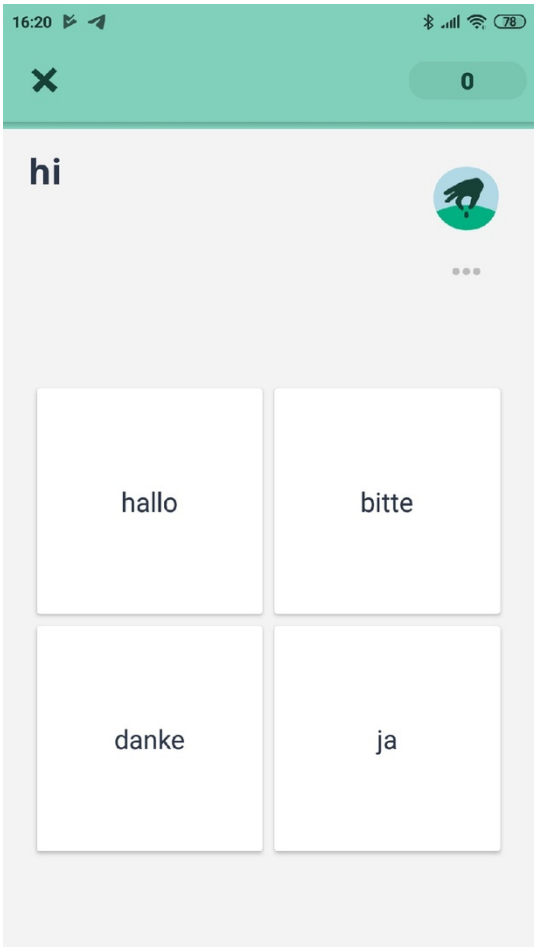

Figure 1.4: Memrise exercise

language.

1.3.4 Summary

Of course, there are many more language learning applications on the market than discussed in this chapter. However, the overwhelming majority of them are either less successful copies of the considered applications or are too different from the goals of my application.

As can be seen, all of them lack some necessary features. Besides, some of them have performance problems, user interface, or material quality. During the development, I will try to take these moments into account and take only the best from these applications.

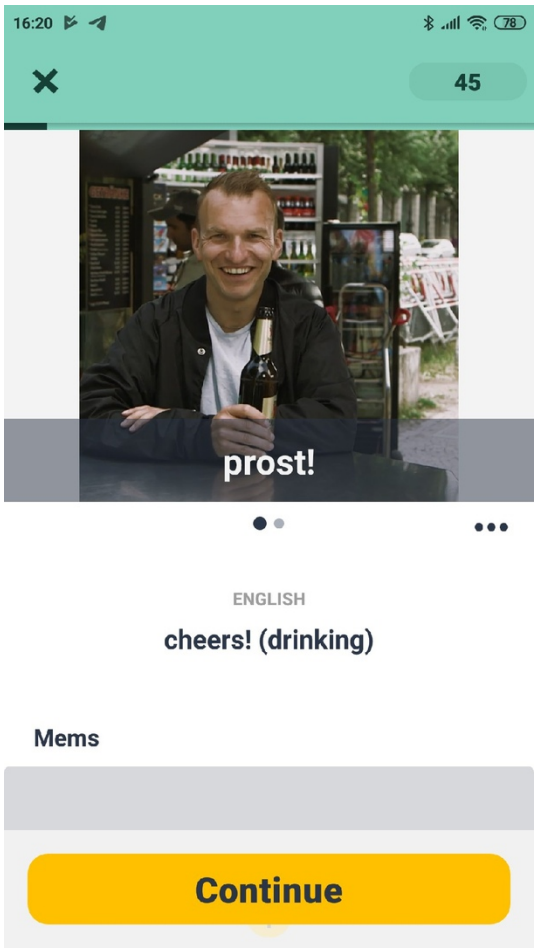

Figure 1.5: Memrise

Figure 1.6: Lingualeo trainings

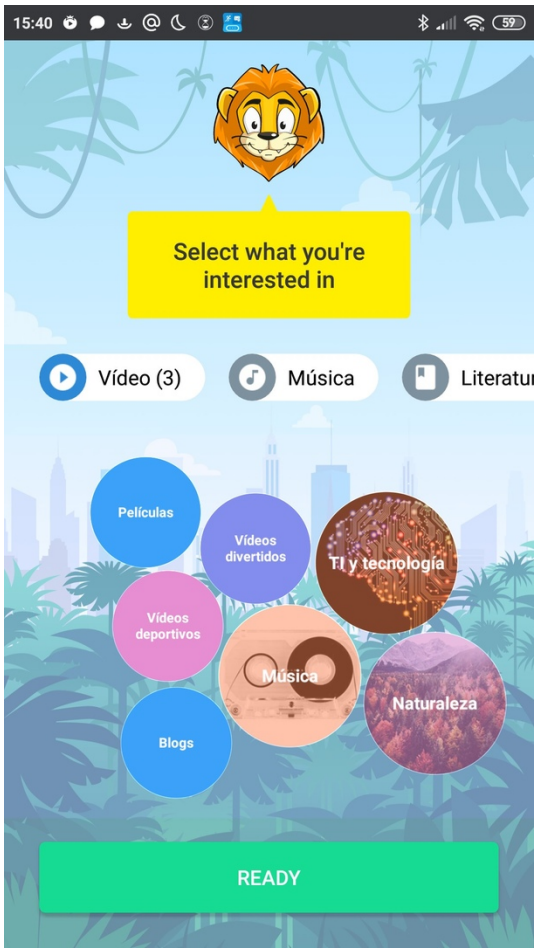

Figure 1.7: Lingualeo topics

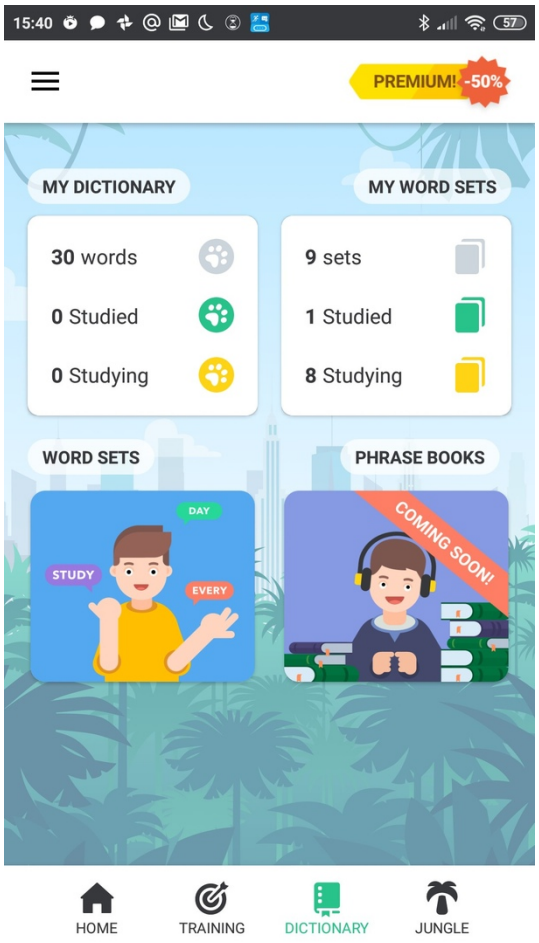

Figure 1.8: Lingualeo user progress

1.4 Handwriting Technologies

The main goal of the application is to bring the experience of solving exercises to the exercises from the textbook as close as possible. Therefore, the choice of handwriting technology is the most crucial task.

So although there are several handwritten applications on the market, I was limited by the fact that text recognition must be carried out without third-party applications. Therefore, there are not so many options:

1. Input using Android Gestures.
2. Google Handwriting Input.
3. Use a recognition system based on OCR (Optical Character Recognition).
4. Use third-party API for recognition of the entered text.

1.4.1 Android Gestures

Pros:

- Part of the Android system
- Easy to implement

Cons:

- Input is possible only by one letter
- Each letter needs to be drawn manually; therefore recognition accuracy is rather low
- No language is supported “out of the box” - you must manually draw the alphabet
- Because these are simple gestures, it is impossible to correct the answer
- Expansion of functionality to meet functional requirements will require a lot of effort and time

1.4.2 Google Handwriting Input

Pros:

- No need to integrate into the application - works out of the box
- High accuracy and recognition speed
- Works without internet

- Supports about 100 languages

Cons:

- It is a separate application - the user will have to download this keyboard to use it with my application.
- Because there is no possibility of integration into my application, this option does not meet the functional requirements

1.4.3 OCR

Pros:

- In theory, the exceptionally high recognition accuracy of almost any handwriting

Cons:

- Implementation of such a recognition system is an extraordinarily complex and voluminous task that goes far beyond the scope of this thesis
- Even existing OCR solutions (for example, Google's Mobile Vision Text API) require fine tuning and are challenging to implement
- This technology is either demanding on mobile device resources (which does not guarantee stable operation on all devices), or a permanent Internet connection. However, more often it requires both the first and second

1.4.4 Third-party APIs

Pros:

- The relative ease of implementation
- There is an open source solution

Cons:

- The vast majority of open source solutions are outdated (for example, Lipi TK was last updated in 2013 and supports Android 4.0.1 and later)
- Supported and developing solutions usually cost money, and free functionality is limited (limit on the number of devices, API calls, the number of processed text, and so on)
- The user needs a constant internet connection for the application to work correctly

1.4.5 Summary

After analyzing the pros and cons of existing solutions, I concluded that recognition with a third-party API is best for my purposes. The product MyScript Interactive Ink was selected.

Pros:

- API described in detail and step-by-step instructions for its implementation in projects Flexible import and export capabilities allow for easy integration into digital environments
- Cross-platform support (Android, iOs, Windows, Web)
- Works stably on tablets and phones, the ability to recognize even on a smartwatches
- Flexible text entry - One or several lines of text can be handwritten
- A limited set of simple gestures makes editing, formatting, and conversion to typeset text easy
- The handwritten text layout dynamically adapts to the device size and orientation, as if it was typeset text
- Supports 65 languages
- The API is continuously updated and refined.

Cons:

- Paid product with a monthly subscription. The free version has a limit of 100 devices and up to 2000 calls per month.

It is also worth mentioning that iInk supports recognition of not only texts but also mathematical formulas and various graphics (diagrams, geometric figures), which may be useful for extending the current application.

1.5 Requirements Specification

1.5.1 Functional Requirements

The application should have the following functionality.

- **F1: Display exercises**
  The application should be able to display grammar exercises on Android device correctly. Completing these exercises should be possible.
  Priority: high
  Difficulty: high

- **F2: Download exercises from server**
  The application should be able to download exercises from a remote server.
  Priority: medium
  Difficulty: medium

- **F3: Add downloaded exercises**
  The application should be able to add downloaded exercises to the application database.
  Priority: medium
  Difficulty: low

- **F4: Review of quiz**
  The application should be able to display a completed quiz overview and show the user's correct and incorrect answers.
  Priority: low
  Difficulty: medium

- **F5: Support three types of exercises**
  The application shall support three types of exercises: full sentence translation, filling a correct verb form to sentence and QA linking.
  Priority: high
  Difficulty: high

- **F6: Track, display, and clear user statistics**
  The application shall track user statistics of completed exercises and show it in the user profile. The user should also be able to delete his statistics.
  Priority: low
  Difficulty: low

1.5.2 Non-functional Requirements

In this section are described requirements that about quality, platform, performance and how the system work, rather than specific instructions on how the application should work.

- **N1: User-friendly and convenient UI**
  The application must have a clear and intuitive user interface.
  Priority: high
  Difficulty: medium

- **N2: Android devices support**
  The application must be natively supported on the majority of Android smartphones and tablets with Android 5.0 or newer.
  Priority: medium
  Difficulty: medium

- **N3: Availability**
  The application should be able to display exercises, user progress, and history as long as the device works correctly. Internet is needed only for handwriting input.
  Priority: high
  Difficulty: low

- **N4: Performance**
  The application should have a responsive and smooth UI without any lags.
  Priority: medium
  Difficulty: low

- **N5: Supported languages**
  The application UI should support Czech, English and Russian languages.
  Priority: low
  Difficulty: low

- **N6: Communication with the server**
  If the implementation of functional requirements demands a server implementation, the application and the server should communicate through HTTP using REST interface.
  Priority: high
  Difficulty: medium

- **N7: Vertical orientation**
  The app will always be used in a vertical orientation.
  Priority: low
  Difficulty: low

1.6 Use Cases

In 1986, Ivar Jacobson, later co-author of the Unified Modeling Language (UML) and the Rational Unified Process (RUP), first formulated a visual modeling technique for describing use cases. During the 1990s, usage scenarios became one of the most common techniques for documenting functional requirements, especially in the object-oriented environment, from where they originated. However their use is not limited to object-oriented systems, since usage scenarios are not object-oriented in nature. [9]

Use case - in software development and system design, this is a description of the behavior of a system when it interacts with someone or something from the external environment. The system can respond to external requests, or it can initiate interaction by itself. In other words, the use case describes “who”

and "what" can do with the system in question, or what the system can do with "who" or "than."

Each use case should focus on describing how to achieve the goal or objectives. The use case defines the interactions between external agents and the system aimed at achieving the goal. An actor is a role played by a person or thing interacting with the system.

In general, the use case should:

- Describe exactly what the system should do to so the actor achieved his goal.
- Do not affect implementation details.
- Have a sufficient level of detail.
- Do not describe user interfaces and screens. This is done during the user interface design.

For a better understanding of the objectives and scope of the project, use cases are often accompanied by text descriptions - scenarios. Usually, use case has a main script and several alternatives.

Now that we know what a use case is, we can design use cases based on functional requirements and describe their flow using scripts

- **UC1: Start the quiz**
  The user will be able to select a quiz type and launch it. There are three possible scenarios.

  First scenario:

  1. The user chooses the "Whole Sentence Quiz."
  2. The application launches a quiz and displays the first whole sentence exercise.

  Second scenario:

  1. The user chooses the "Fill-In Quiz".
  2. The application launches quiz and displays first fill-in exercise.

  Third scenario:

  1. The user chooses the "Matching Quiz".
  2. The application launches quiz and displays first matching exercise.

- **UC2: Complete Whole Sentence exercise**

1. The actor should be able to view and complete Whole Sentence exercise.
2. The application displays a Whole Sentence exercise.
3. The actor completes the exercise by writing the answer in a special field.
4. The application checks the actor's answer and depending on whether it was correct or not two scenarios are possible.

First scenario:

1. The answer is correct.
2. The application shows the actor that his answer is correct.

Second scenario:

1. The answer is wrong.
2. The application shows the actor that his answer is wrong and shows the correct answer.

- **UC3: Complete Fill-In exercise**

1. The actor should be able to view and complete Fill-In exercise.
2. The application displays a Whole Sentence exercise.
3. The actor completes the exercise by writing the answer in a special field.
4. The application checks the actor's answer and depending on whether it was correct or not two scenarios are possible.

First scenario:

1. The answer is correct.
2. The application shows the actor that his answer is correct.

Second scenario:

1. The answer is wrong.
2. The application shows the actor that his answer is wrong and shows the correct answer.

- **UC4: Complete Matching exercise**

1. The actor should be able to view and complete Matching exercise.
2. The application displays a Matching exercise.

3. The actor completes an exercise by connecting the corresponding objects with a line.
4. The application checks the actor's answer and depending on whether it was correct or not two scenarios are possible.

First scenario:

1. The answer is correct.
2. The application shows the actor that his answer is correct.

Second scenario:

1. The answer is wrong.
2. The application shows the actor that his answer is wrong.

- **UC5: Review quiz results**

1. The application will be able to show the actor information about completed quiz.
2. The actor completes the final quiz exercise. Depending on the type of quiz, two scenarios are possible.

First scenario:

1. The quiz type is Whole Sentence or FI.
2. The application shows a complete quiz with all questions, the actor's answers, and correct answers.

Second scenario:

1. The quiz type is MA.
2. The application shows the number of correct and wrong answers.

- **UC6: Display profile**

1. The application shows the profile of the actor with his statistics and points.
2. The actor chooses the "Profile" option.
3. The application displays the actor's profile.

- **UC7: Display settings**

1. The application shows application settings.
2. The actor chooses the "Settings" option.
3. The application displays application settings.

- **UC8: Display quizzes**

  1. The application shows quizzes with three different quiz types.
  2. The actor chooses the “Quizzes” option.
  3. The application displays quizzes types.

- **UC9: Download exercises from server**

  1. The actor should be able to download exercises from the server.
  2. The actor chooses the “Settings” option.
  3. The application displays application settings.
  4. The actor chooses the “Get new exercises” option. Depending on the success of this operation, two scenarios are possible.

  First scenario:

  1. Exercises successfully downloaded from the server.
  2. The application shows a “Success” message.

  Second scenario:

  1. Failed to download exercises from the server.
  2. The application shows “Error” message.
  3. Four

- **UC10: Clear user statistics**

  1. The actor should be able to clear the user's statistics.
  2. The actor chooses the “Settings” option.
  3. The application displays application settings.
  4. The actor chooses the “Reset your statistics” option.
  5. The application shows a dialogue with two options. Three scenarios are possible.

  First scenario:

  1. The actor chooses the “OK” option.
  2. The application clears user statistics.
  3. The application shows a “Success” message.

  Second scenario:

  1. The actor chooses the “Cancel” option.
  2. The application closes the dialog.

	F1	F2	F3	F4	F5	F6
U1	■					
U2	■				■	
U3	■				■	
U4	■				■	
U5				■		
U6						■
U7		■				
U8	■					
U9		■	■			
U10						■

Table 1.2: Functional Requirements covered by Use Cases

Third scenario:

1. The actor presses the back button on the device.
2. The application closes the dialog.

1.6.1 Summary

Functional requirements describe the system as a whole, and use cases describe the system from the user's point of view. Moreover,although these approaches contain different levels of detail about the system, they describe the same system. Therefore, they should completely cover each other. The Table 1.2 shows how the use cases described covers the functional requirements.

1.7 Exercises Database

For the user to perform quizzes and exercises, it is necessary that these exercises be present in the application. There are several possible ways to collect

and populate an exercise database. Consider them on the example of exercises for learning English.

1.7.1 Made by someone Database with Exercises

Although this option looks the most convenient, I never managed to find a well-organized database with exercises. Perhaps I was looking poorly, but I got the impression that such databases are not in the public domain.

1.7.2 Third Party APIs

It is possible to connect to sites with dictionaries that have an open API and use it to fill the database with exercises.

Similar sites exist, for example, Dictionary.com. However, there are problems with the implementation - besides the fact that the exercises need to be brought to a format that the developed application understands, it is not entirely clear where to implement an API call.

1. **Call on the application side.**
   In this case, there is practically no need for a private server, since it will only store information about users and their statistics. It can also lead to an excessive increase in the database within the application.

2. **Call on the server side.**
   This option requires considerable effort and time and complicates the server architecture.

Due to the reasons described above, this option is far beyond the scope of this thesis.

1.7.3 Make Exercises from Corpus

It is possible to take a significant corpus with sentences, parse and process it and generate exercises based on it in the right format to populate the database.

However, a suitable corpus is not so easy to find, then it will be necessary to write a proper parser and an exercise generator for it.

1. For the type of exercise Whole Sentence, you must somehow find the correct translation of the sentence or make a machine translation. However, in this case, the quality of the exercises can suffer greatly.

2. For the type of exercise Fill-In, it is necessary to correctly determine which word to check and then select the appropriate neutral form for it.

3. For Matching exercise type, this filling method is not applicable.

The disadvantage of this method is also that it is practically impossible to sort the resulting exercises by difficulty, category, and so on.

Therefore, this option is also not suitable.

1.7.4 Hand-crafted Exercises

The easiest way to add exercises to the database is that the exercises will be immediately in a convenient format; you can assign them a specified complexity, categories, and other parameters.

However, it is evident that using this method to generate any significant number of exercises requires a tremendous amount of time and effort.

Due to its simplicity and convenience, this method was chosen for the initial filling of the database with exercises. These exercises the application contains immediately "out of the box", so that the user has something to do until he receives the exercises in other ways.

1.7.5 User-created Exercises

You can allow users to invent and add their exercises to the database. This is not the easiest to implement solution, which, if implemented in an application, would require the creation of an exercise constructor. However, with a large user base in this way you can get a large number of potentially high-quality exercises.

It was done for example in the Memrise application and worked more or less successfully. However, about a year ago they removed the ability to add custom courses.

This method is a good option for further expansion. However, the creation of an exercise constructor is beyond the scope of this work. However,the ability to add exercises will be present in the final API of the developed server.

1.7.6 Summary

Unfortunately, there is no simple solution, and all the above options have serious drawbacks. As part of this work, manually created exercises will be added to the applications, and a server API will be developed that supports adding exercises to the database.

Chapter 2

Design

2.1 Domain Model

2.2 Android Application

This chapter will cover the user interface and data storage design for the application. When discussing the goals of the project, much attention was given to the convenience and beauty of the user interface. Therefore, this aspect will be given much attention.

2.2.1 UI Design

The user interface is one of the most critical factors for the success of an application since, for the user, this is the main way to understand what the application is made for and how to use it. If the UI is ugly and inconvenient, it will be difficult for users to use the application by the usage scenario and various errors may occur. A bad design with a high probability will push the user away from the application.

It is essential to understand the needs and motivation of the people who will use the resulting product for developing an application. Next, domain restrictions and technical requirements should be taken into account. Together with the previous paragraph provides the basis for the development of user interface design. However, above all, the design should be designed for people who will use this product.

2.2.1.1 Main UI Principles

When developing a UI design, it is recommended to adhere to the following rules [10]:

1. Accessibility
   Systems should be designed to be usable, without modification, by as

many people as possible.

2. Aesthetically Pleasing
   Provide visual appeal by following these presentation and graphic design principles:
   - Provide meaningful contrast between screen elements.
   - Create groupings.
   - Align screen elements and groups.
   - Provide three-dimensional representation.
   - Use color and graphics effectively and simply.
3. Availability
   - Make all objects available at all times.
   - Avoid the use of modes.
4. Clarity
   The interface should be visually, conceptually, and linguistically clear including:
   - Visual elements
   - Functions
   - Metaphors
   - Words and text
5. Configurability
   Permit easy personalization, configuration, and reconfiguration of settings to do the following:
   - Enhance a sense of control.
   - Encourage an active role in understanding.
6. Consistency
   A system should look, act, and operate the same throughout. Similar components should:
   - Have a similar look.
   - Have similar uses.
   - Operate similarly.

   The same action should always yield the same result. The function of elements should not change. The position of standard elements should not change.

7. Directness
   Provide direct ways to accomplish tasks.

   - Available alternatives should be visible.
   - The effect of actions on objects should be visible

8. Efficiency
   Minimize eye and hand movements, and other control actions.

   - Transitions between various system controls should flow easily and freely.
   - Navigation paths should be as short as possible.
   - Eye movement through a screen should be obvious and sequential.

   Anticipate the user's wants and needs whenever possible.

When developing the user interface, I was following the rules described above, the official Android guidelines and knowledge gained on the subject of the master's program User Interface Design (Czech Technical University in Prague, Faculty of Information Technology) [11]

2.2.1.2 Transition Diagram

In previous chapters, the requirements and essential functions of the application were described in detail, along with the Use Cases. Based on this analysis, a user interface can be designed.

The application should have three main screens:

- **Quizzes** - selection and launch of a quiz, also serves as the main screen of the application.
- **Profile** - a user profile page where user can see his statistics.
- **Settings** - screen with application settings. Here the user can change his profile settings and download or download new exercises from the server. Also here sits some basic information about the application like authors and license.

The next important part is the screens of different quizzes, where the user can solve the exercises. By the assignment, three such screens were created: Whole Sentence Quiz, Fill-In Quiz, and Matching Quiz.

Transition diagram is shown on Figure 2.1.

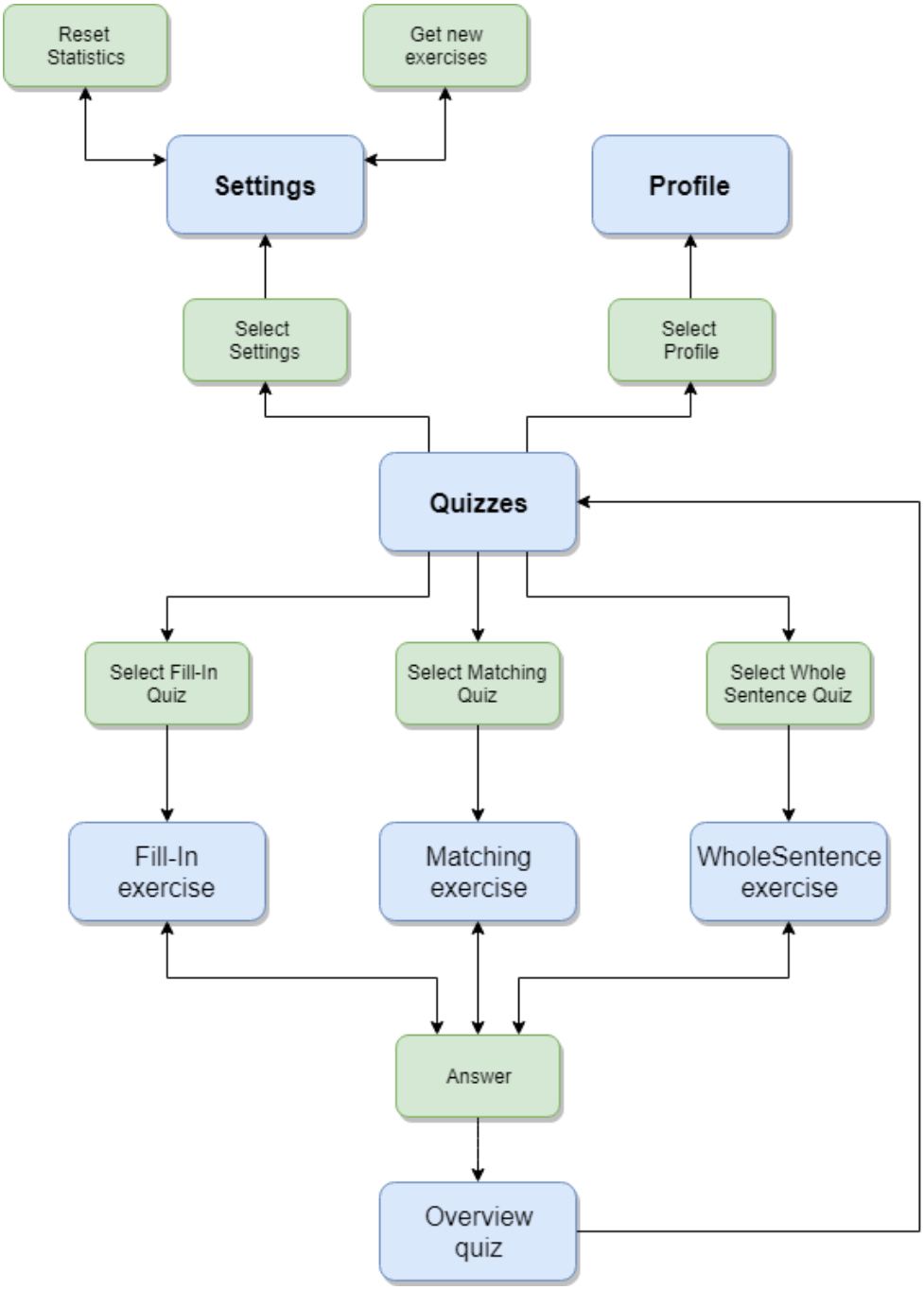

Figure 2.1: Transition Diagram

2.2.1.3 Lo-fi Prototype

The next step is to design a lo-fi application prototype that will take into account the requirements and rules mentioned above. Based on this prototype, then it will be possible to create a hi-fi prototype of the application.

This prototype is quite simple. It was used to understand the concept and roughly estimate which elements will be located on which screen.

On Figure 2.2, Figure 2.3, and Figure 2.4 the main Lo-Fi prototype screens are shown.

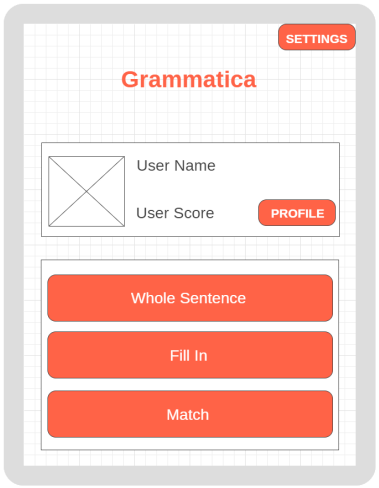

Figure 2.2: Lo-Fi Main Screen

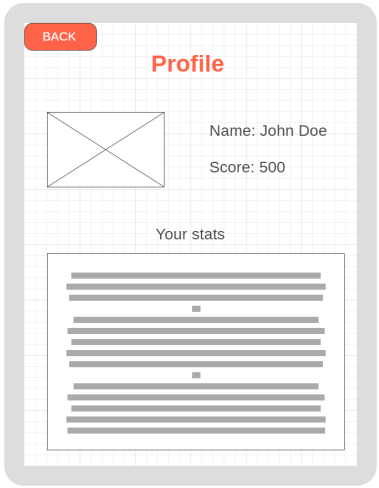

Figure 2.3: Lo-Fi Profile Screen

Figure 2.4: Lo-Fi Exercise Screen

2.2.1.4 Hi-fi Prototype

Hi-fi prototype should reflect the final design of the application and take into account the analysis of the previous lo-fi prototype. The final Android application should follow the concepts of Material Design and use the elements that users of this OS are used to.

When testing and analyzing the lo-fi prototype for compliance with the principles from the Chapter 2.2.1.1, a significant drawback was found - substantial congestion of the main screen of the application. There are too many buttons and functions. Therefore, following the principles of Availability and Efficiency, the hi-fi prototype was fixed, and the Bottom navigation bar was chosen as the primary navigation method tool for moving between screens.

Also, after additional analysis of the lo-fi prototype and competitors in the market, the method of notifying the user about the correct or incorrect answer to the exercise was finally chosen. The selected method is a dialogue that notifies the user about the correct answer and allows him to proceed to the next question. In case of an incorrect answer, the user is not able to correct it, and the correct answer is shown in the dialogue.

On Figure 2.5, Figure 2.6, and Figure 2.7 the main Hi-Fi prototype screens are shown.

2.2.1.5 Design Evaluation

10 Usability Heuristics for User Interface Design by Jakob Nielsen [12] were used for heuristic evaluation. The Grammatica app has been examined from 10 different points of view.

Figure 2.5: Hi-Fi Main Screen

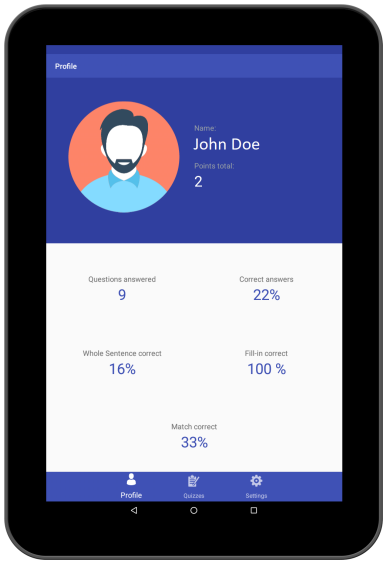

Figure 2.6: Hi-Fi Profile Screen

1. Visibility of system status.
   The system should always keep users informed about what is going on, through appropriate feedback within reasonable time.

   The application responds to user actions without delay; switching between screens occurs almost instantly.

   In the future, it is worth adding also the Progress bar for all operations

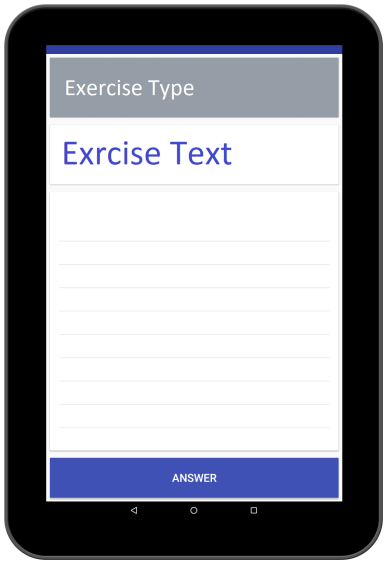

Figure 2.7: Hi-Fi Exercise Screen

of communication with the server, since they can potentially take much time due to internet problems.

2. Match between system and the real world.
   The system should speak the users' language, with words, phrases and concepts familiar to the user, rather than system-oriented terms. Follow real-world conventions, making information appear in a natural and logical order.

   The application does not contain so many icons, but each intuitively makes it clear for what action it is responsible.

   It is also worth adding icons to the Settings screen so that the user can quickly understand what each section of settings is responsible for.

3. User control and freedom.
   Users often choose system functions by mistake and will need a clearly marked "emergency exit" to leave the unwanted state without having to go through an extended dialogue. Support undo and redo.

   Android has a standard back button, so this requirement is fulfilled.

4. Consistency and standards
   Users should not have to wonder whether different words, situations, or actions mean the same thing. Follow platform conventions.

   Except for the Matching Exercise screen, the application uses standard components of the Android platform and therefore is similar to most other Android applications.

5. Error prevention
   Even better than good error messages is a careful design which prevents a problem from occurring in the first place. Either eliminate error-prone conditions or check for them and present users with a confirmation option before they commit to the action.

   At the moment, the application has one action that requires confirmation. When this action is selected, a dialog appears, and the user can cancel or confirm the operation.

   In the future, confirmatory dialogues will be added for all important profile and exercise operations.

6. Recognition rather than recall
   Minimize the user's memory load by making objects, actions, and options visible. The user should not have to remember information from one part of the dialogue to another. Instructions for use of the system should be visible or easily retrievable whenever appropriate.

   From this point of view, the application is as simple as possible - no screen contains any additional menus or hidden buttons.

7. Flexibility and efficiency of use.
   Accelerators — unseen by the novice user — may often speed up the interaction for the expert user such that the system can cater to both inexperienced and experienced users. Allow users to tailor frequent actions.

   There are currently no advanced features in the application that will require user training. This requirement is not applicable.

8. Aesthetic and minimalist design
   Dialogues should not contain information which is irrelevant or rarely needed. Every extra unit of information in a dialogue competes with the relevant units of information and diminishes their relative visibility.

   The application contains only the necessary information for the user. This requirement is met.

9. Help users recognize, diagnose, and recover from errors
   Error messages should be expressed in plain language (no codes), precisely indicate the problem, and constructively suggest a solution.

   Currently, there is only one type of task in the application where an error message is required - interaction with the server. The error appears as a pop-up message and does not contain any codes. However, the error message does not depend on the type of error and offers no solution. It should be fixed in the future.

Rule	Requirement met	Solution
1	No	Add progress bar for operations with the server.
2	Yes	Add icons to Settings screen.
3	Yes	—
4	Yes	—
5	Yes	—
6	Yes	—
7	Yes	—
8	Yes	—
9	No	Add an error message with clear error cause and possible solution.
10	No	Add user manual.

Table 2.1: Design evaluation results

10. Help and documentation
    Even though it is better if the system can be used without documentation, it may be necessary to provide help and documentation. Any such information should be easy to search, focused on the user's task, list concrete steps to be carried out, and not be too large.

    No user manual is available at the moment. Moreover, although the application is quite simple, this instruction should be added.

The results of the analysis are presented in the Table 2.1.

2.2.2 Application Data Storage

Android has three basic ways to store data on a device.

2.2.2.1 Shared Preferences

The easiest and fastest way to store a small amount of information like user settings. Data is stored as a key-value pair and is limited to strings and primitive data types only.

This method is going to be used to store user statistics. In the future, when the ability to save statistics on the server is added, it makes sense to switch to the SQLite database.

2.2.2.2 Device Storage

Each Android application has its own internal storage directory that interacts with it, in which the application can store text and binary files. Files within this directory are not accessible by the user or other applications installed on the user's device. They are also automatically deleted when the user deletes the application. Interaction with such files occurs using the built-in File API.

It is also possible to store data in external storage, but for this, the application needs to obtain user permission and external storage must be accessible (mounted).

2.2.2.3 SQLite Database

Each Android application can create and use SQLite databases for storing large amounts of structured data. SQLite is not only relatively light, but also very fast. This method will be used to store all the exercises on the device.

2.2.2.4 Other Resourcecs

Application images, constants, colors, styles, strings are contained in application resources. It is a special folder that is packaged in an installation file at the project assembly stage. Resources are sent to the same folder for specific device configurations (for example, application translations). During startup, the system determines the device configuration and automatically loads the appropriate resources.

Following the functional requirements, String resources will be available in English, Czech and Russian. The application will determine which language to use based on the language of the mobile device system.

2.3 Server Design

This chapter shows the API interface design for client-server communication and data exchange.

2.3.1 REST

REST (Representational State Transfer) is a software architecture style for distributed systems, such as the World Wide Web, which is typically used to build web services. REST was designed to make optimal use of the HTTP protocol, which is its main benefit.

Systems that support REST are called RESTful systems. In general, REST is a straightforward information management interface.

Access to the resource on the server should be provided via the URL and support four basic operations - Create, Read, Update, Delete. The server responds to the call with standard status codes.

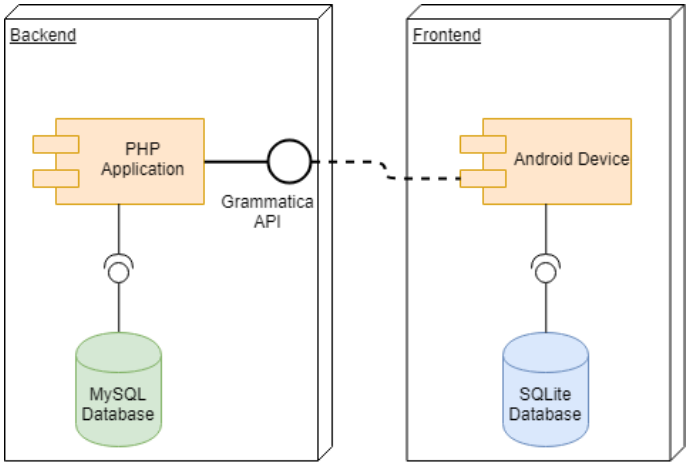

Figure 2.8: System Architecture

2.3.2 Requests

This chapter will describe in detail the data exchange between the client and the server. The exchange takes place in JSON format using the specified URL. Their overview, along with HTTP operations, is provided in Table 2.2 at the end of this chapter.

The default root URL is `http://localhost/grammatica/v1/`. All operations except registration and login require user authorization.

2.3.2.1 User Registration

- **Method:** POST
- **URI:** register
- **Parameters:** name, email, password

New User Registration. The server will check for the uniqueness of the user's email.

2.3.2.2 User Login

- **Method:** POST
- **URI:** login
- **Parameters:** email, password

Authorize an existing user. In response, the user receives his unique API key, which must be used to authorize all operations except login and registration.

2.3.2.3 Create new Exercise

- **Method:** POST
- **URI:** exercises
- **Parameters:** type, exercise

Creating a new exercise. You must specify the type of exercise ("wh," "fi" or "ma") and the exercise itself in the form of JSON. An example of the exercise field can be found below.

```
{
"exercises": [
 {
   "id": "0",
   "type": "wh",
   "exercise": [
       {
         "question": "Ahoj?",
         "answer": "Hi"
       }
]},
 {
   "id": "1",
   "type": "fi",
   "exercise": [
       {
         "question1": "The match?",
         "bracket": "start",
         "question2": "in 5 minutes.",
         "answer": "starts"
       }
]}]}
```

Listing 2.1: Exercises example

2.3.2.4 Get all User Exercises

- **Method:** GET
- **URI:** exercises
- **Parameters:** -

Returns all user exercises. The method only requires the presence of an API key in the header.

2.3.2.5 Get a Single User Exercise

- **Method:** GET
- **URI:** exercises/id
- **Parameters:** -

Returns specific user exercise. As above, the method only requires the presence of an API key in the header.

2.3.2.6 Update a Single User Exercise

- **Method:** PUT
- **URI:** exercises/id
- **Parameters:** type, exercise

Updating single user task. The requirements are the same as for creating a new exercise.

2.3.2.7 Delete Single User Exercise

- **Method:** DELETE
- **URI:** exercises/id
- **Parameters:** -

Removes a specific user exercise. The method only requires the presence of an API key in the header.

2.3.3 Responses

The server will use the standard HTTP response codes.

Each response includes an additional error message about the success of the operation.

The following are examples of server responses with corresponding requests from the client.

URL	Method	Parameters	Description
grammatica/v1/register	POST	name, email, password	User registration
grammatica/v1/login	POST	email, password	User login
grammatica/v1/exercises	POST	type, exercise	Create a new exercise
grammatica/v1/exercises	GET	—	Get all exercises
grammatica/v1/tasks/id	GET	—	Ger exercise
grammatica/v1/tasks/id	PUT	—	Update exercise
grammatica/v1/tasks/id	DELETE	—	Delete exercise

Table 2.2: API methods overview

2.3.3.1 User Login

```
1 POST /login HTTP/1.1
2 HOST: localhost
3 content-type: application/x-www-form-urlencoded
4 content-length: 33
5 email=test@mail.com&password=test
6 {
7         "error": false,
8         "name": "Test",
9         "email": "test@mail.com",
10        "apiKey": "
              a56741cc8664f0ce89a48032defcef0e",
11        "createdAt": "2019-05-07 00:20:09"
12 }
```

Listing 2.2: User login response

2.3.3.2 Create User Exercises

```
1 POST /exercises HTTP/1.1
2 HOST: localhost
3 content-type: application/x-www-form-urlencoded
4 authorization: a56741cc8664f0ce89a48032defcef0e
5 content-length: 40
6 type=wh&exercise={"question": "What is your name
    ?","answer": "Jak se jmenujes?"}
7 {
8         "error": false,
9         "message": "Exercise created successfully
              ",
10        "exercise_id": 3
11 }
```

Listing 2.3: Create exercise response

2.3.3.3 Get User Exercises

```
GET /exercises HTTP/1.1
HOST: localhost
content-type: application/x-www-form-urlencoded
authorization: a56741cc8664f0ce89a48032defcef0e
{
"error": false,
"exercises": [
{
        "id": 1,
        "type": "wh",
        "exercise": "{ "question": "Ahoj", "answer
           ": "Hi" }",
        "status": 0,
        "createdAt": "2019-05-07 18:24:42"
},
{
        "id": 2,
        "type": "wh",
        "exercise": "{ "question": "What is your
          name?", "answer": "Jak se jmenujes?"
          }",
        "status": 0,
        "createdAt": "2019-05-07 18:26:32"
}],}
```

Listing 2.4: Get exercises response

2.3.3.4 Get User Exercises without API Key

```
GET /exercises HTTP/1.1
HOST: localhost
content-type: application/x-www-form-urlencoded
{
        "error": true,
        "message": "Api key is misssing"
}
```

Listing 2.5: Get exercises without API response

2.3.3.5 Get User Exercises wrong API Key

```
1 GET /exercises HTTP/1.1
2 HOST: localhost
3 content-type: application/x-www-form-urlencoded
4 authorization: dfdffgfg
5 {
6         "error": true,
7         "message": "Access Denied. Invalid Api key
              "
8 }
```

Listing 2.6: Get exercises wrong API response

2.3.4 Server Database Architecture

The server must store information about users and their exercises. For these purposes, the database shall be used. A conceptual model can be seen on Figure 3.2.

2.3.5 Authorization and User Account Security

To fulfill the functional requirements, it is necessary to implement the ability to authorize users. For this, third-party services can be used, like OAuth, but I decided to limit myself to simple key generation for each user. With this key, the user identifies himself and can conduct operations with his own resources.

Chapter 3

Implementation

3.1 Android client implementation

The main focus of this thesis is the implementation of an application for the Android operating system. Therefore, for the implementation of the frontend part of the system was allocated the largest amount of time and effort. The following chapters will describe the most essential and exciting implementation details. Development was carried out in the Android Studio IDE.

3.1.1 Tools and libraries

3.1.1.1 Language and IDE

In general, applications for the Android operating system are mostly written in the Java programming language using the Android SDK. It is also possible to develop applications for Android using other languages (for example, Kotlin). However, for this project, Java was chosen as the more common and straightforward method.

3.1.1.2 Android support library

Android Support Libraries is a collection of libraries designed to ensure the backward compatibility of new API versions with old Android versions. Because the minimum API version for Grammatica is 20, in many places these libraries were necessary to use.

3.1.1.3 Retrofit

To understand the benefits of Retrofit, it is vital to figure out how to send a network request using only the Android SDK. Network operations cannot be performed in UI thread. Therefore, it is obligatory to use AsyncTask to create another thread, then `HttpsUrlConnection` for the network call, and

then notify the user about the progress through AsyncTask. Moreover, the data obtained with InputStream, which means that it is should be converted to some more useful format. Also, it is needed to correctly handle the re-creation and destruction of the Activity, because otherwise, it will cause a configuration change. Moreover, these are only the most basic points. As a result, to implement a simple network call, the developer has to write a lot of code.

Retrofit, a prevalent library for network communication in Android programming, will help us to avoid this whole process. This library is even mentioned as a recommended library in the Google official development guide for Android [13]. This library perfectly supports the REST API, is easy to integrate and test. Retrofit establishes a connection, creates HTTP requests, sends data handles the callbacks.

To integrate the library, it is necessary to create a model class and an interface. An example of implementation will be given below.

3.1.1.4 GSON

Retrofit by default can deserialize the HTTP bodies into a ResponseBody type and accept only the RequestBody type; the GSON library will be used for deserialization and serialization purposes. This is a library for serialization (deserializing) Java objects to (from) JSON representations.

3.1.1.5 MyScript Interactive Ink SDK

Iink SDK is a development toolkit provided by MyScript to help integrators build Interactive-Ink-enabled applications. This library is needed for the implementation of one of the main features of the application - handwriting support.

3.1.2 UI implementation

As is known from the previous chapters, the application has three main screens - Quizzes, Profile, and Settings. They are implemented using three Fragment classes (`QuizeFragment`, `ProfileFragment`, `SettingsFragment`), each of which is nested inside `BaseActivity`. Moving between fragments is implemented using the `BottomNavigationView`. Thus, `BaseActivity` contains only `BottomNavigationView` and `FrameLayout`, which is a container for an individual Fragment.

```
1 <android.support.design.widget.CoordinatorLayout
      xmlns:android="http://schemas.android.com/apk/
      res/android"
2       xmlns:app="http://schemas.android.com/apk/res-
            auto"
3       xmlns:tools="http://schemas.android.com/tools"
4       android:id="@+id/containet"
5       android:layout_width="match_parent"
6       android:layout_height="match_parent"
7       tools:context=".Activities.BaseActivity">
8
9       <FrameLayout
10          android:id="@+id/frame_container"
11          android:layout_width="match_parent"
12          android:layout_height="match_parent"
13          app:layout_behavior="@string/
                appbar_scrolling_view_behavior"
14          android:layout_marginBottom="80dp" />
15
16      <android.support.design.widget.
            BottomNavigationView
17          android:id="@+id/navigationView"
18          android:layout_width="match_parent"
19          android:layout_height="wrap_content"
20          android:layout_gravity="bottom"
21          android:background="@color/colorPrimary"
22          app:itemBackground="@color/colorPrimary"
23          app:itemIconTint="@color/bottom_nav_color"
24          app:itemTextColor="@color/bottom_nav_color
                "
25          app:menu="@menu/navigation" />
26
27 </android.support.design.widget.CoordinatorLayout>
```

Listing 3.1: BaseActivity layout

Also very important part of the application are 3 screens with exercises - one for each type of exercise `ExerciseWholeSentenceActivity`, `ExerciseFillinActivity`, `ExerciseMatchActivit`)). They were also planned to be implemented using fragments, but in this case, elusive errors occurred because of the iink SDK and the application stopped working stably. An issue for the support team of this SDK was created, but no response was received yet. Due to these factors, these screens are implemented using the Activity classes.

The text recognition system is used in two types of exercises - the Whole Sentence and Fill-In. For the Matching exercise, a custom View `ExerciseMatchLineView`

was implemented.

Dialogues about the correctness of the user response (`dialogCorrect` and `dialogWrong`) are implemented using `BottomSheetDialog`.

A dialog with an overview of a completed quiz (dialogExerciseOverview) is a Fragment with a RecyclerView inside. The RecyclerView contains the results of the completed quiz, and it is filled in using the custom DialogOverviewRecyclerViewAdapter.class, which extends `RecyclerView.Adapter<DialogOverviewRecyclerViewAdapter.ViewHolder>`. A layout for a separate RecyclerView item could be found in itemExerciseOverview.

Screenshots of the implemented application can be found in Appendix B.

3.1.3 Handwriting input

This chapter describes the implementation of handwriting input using the iink SDK. First, the most important aspects necessary for the correct integration of this SDK will be briefly described. Then will follow the implementation examples taken from the Grammatica application.

3.1.3.1 Certificate

To use this API, it is necessary to get a certificate. It is a binary key, which can be obtained by registering on MyScript Developer Portal [14]. The certificate comes embedded in a source file that should be included in the project.

3.1.3.2 Engine

The iink SDK runtime is represented by an Engine object, which can be instantiated with create() method, where MyCertificate - previously obtained certificate.

```
1  public class IInkApplication extends Application {
2      private static Engine engine;
3
4      public static synchronized Engine getEngine()
           {
5          if (engine == null) {
6              engine = Engine.create(MyCertificate.
                   getBytes());
7          }
8          return engine;
9      }
10 }
```

Listing 3.2: IInkApplication class

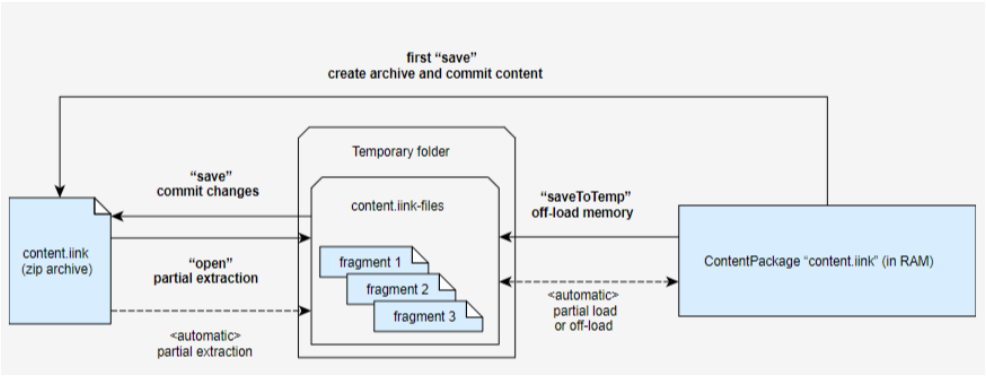

Figure 3.1: Package Diagram

For text recognition, it is required to provide the engine with configuration files (*.conf) that specify required parameters and recognition assets to be loaded.

```
1 Engine engine = IInkApplication.getEngine();
2 Configuration conf = engine.getConfiguration();
3
4 String confDir = "zip://" + getPackageCodePath() +
      "!/assets/conf";
5 conf.setStringArray("configuration-manager.search-
      path", new String[]{confDir});
6
7 String tempDir = getFilesDir().getPath() + File.
      separator + "tmp";
8 conf.setString("content-package.temp-folder",
      tempDir);
```

Listing 3.3: Engine configuration

3.1.3.3 Packages

ContentPackage - is a container storing ink and its interpretation as an ordered collection of parts. It can be saved as a file on the file system and later reloaded or shared between users. Text recognition is not the easiest task and may require many system resources. What is why iink SDK uses the file system to limit the amount of RAM that is required at a given point in time, as well as to store temporary files. Even if the developer does not want to save any content, the SDK will place the files in a temporary folder.

The following Diagram 3.1 explains how iink SDK manages the memory, as well as the role of the different save operations.

3.1.3.4 Parts

ContentPart corresponds to a standalone content unit that can be processed by iink SDK. Each part has a specific type, corresponding to the type of its root block.

3.1.3.5 Renderer

A renderer is a component in charge of deciding how to render the content of each layer, knowing which area of the model needs to be refreshed, as well as parameters such as zoom factor or view offset. It will issue rendering commands, through a canvas object that will do the actual drawing operations.

Each layer can be refreshed independently. Supported layers:

- Background - corresponding for instance to the guides of a Text Document.
- Model - corresponding to everything in the model that was already processed by the engine.
- Temporary - corresponding to temporary elements (like temporary items from drag and drop operation).
- Capture - rendering the ink drawn on the screen but not yet processed by the engine.

3.1.3.6 Editor

An Editor works on a ContentPart, and plays the central role in interaction with content.

3.1.3.7 Handwriting preparation

Inside each screen, where handwriting is needed, there is an EditorView:

```
1          <LinearLayout
2               android:layout_width="match_parent"
3               android:layout_height="match_parent"
4               android:orientation="vertical">
5
6               <include layout="@layout/editor_view"
                    />
7          </LinearLayout>
```

Listing 3.4: Layout with editor view

Then he is assigned with an IEditorListener which monitors the change in the state of the view.

```
1  final Editor editor = editorView.getEditor();
2  editor.addListener(new IEditorListener() {
3  @Override
4  public void partChanging(Editor editor,
       ContentPart contentPart, ContentPart
       contentPart1) {}
5
6  @Override
7  public void partChanged(Editor editor) {
8          invalidateOptionsMenu();
9  }
10
11 @Override
12 public void contentChanged(Editor editor, String[]
       strings) {
13         invalidateOptionsMenu();
14 }
15
16 @Override
17 public void onError(Editor editor, String blockId,
       String message) {
18         Log.e(TAG, "Failed to edit block \"" +
               blockId + "\"" + message);
19 }
20 });
```

Then, the ContentPart is assigned to the Editor, and now it is possible to start working with recognition.

```
1           try {
2               contentPackage = engine.createPackage(
                    file);
3               contentPart = contentPackage.
                    createPart("Text"); // Choose type
                    of content (possible values are: "
                    Text Document", "Text", "Diagram",
                    "Math", and "Drawing")
4           } catch (IOException e) {
5               Log.e(TAG, "Failed to open package \""
                    + packageName + "\"", e);
6           } catch (IllegalArgumentException e) {
7               Log.e(TAG, "Failed to open package \""
                    + packageName + "\"", e);
8           }
9
10          // wait for view size initialization
                before setting part
11          editorView.post(new Runnable() {
12              @Override
13              public void run() {
14                  editorView.getRenderer().
                        setViewOffset(0, 0);
15                  editorView.getRenderer().
                        setViewScale(1);
16                  editorView.setVisibility(View.
                        VISIBLE);
17                  editor.setPart(contentPart);
18              }
19          });
```

3.1.3.8 Results processing

When the Answer button is clicked, it is possible to recognize and obtain user's input. Method waitForIdle() blocks the thread until the engine is idle. So it is called to ensure, that editor will wait for the recognition to complete before exporting any results.

```
1           editor = editorView.getEditor();
2           //Wait for recognition to complete
3           editor.waitForIdle();
4           editor.convert(null, editor.
                getSupportedTargetConversionStates(null
                )[0]);
5           String result = "";
6           try {
7               result = editor.export_(null, MimeType
                    .TEXT);
8           } catch (IOException e) {
9               e.printStackTrace();
10          }
```

Listing 3.5: Processing the result

After the results are obtained, the method `editor.clear()` called to clear the editor for any further use.

3.1.3.9 Shutting down the recognition

After finishing working with text recognition, it is necessary to close all unnecessary instances. Otherwise, they will continue to consume system resources and will cause errors (for example, if the Activity is destroyed but the editor is not). The most convenient way to do this is in the `OnDestroy()` method.

```
1       @Override
2       protected void onDestroy() {
3           editorView.setOnTouchListener(null);
4           editorView.close();
5           if (contentPart != null) {
6               contentPart.close();
7               contentPart = null;
8           }
9           if (contentPackage != null) {
10              contentPackage.close();
11              contentPackage = null;
12          }
13          // IInkApplication has the ownership, do
                not close here
14          engine = null;
15          super.onDestroy();
16      }
```

Listing 3.6: onDestroy method

3.1.4 Database

For the implementation of the exercise database within the application, a tool was selected that is included in each device on Android - Open Source database SQLite. It supports SQL syntax, standard relational data model, transactions, and more. Since this is the standard Android OS tool, the developer does not need to maintain or customize the database operation - it is enough to define the SQL statement to create the database, and from that moment it will be managed by Android system itself. All database operations are implemented inside DBExerciseHelper.class, which extends SQLiteOpenHelper.class.

Moreover, as mentioned earlier, SharedPreferences are used to store user statistics.

3.1.5 Android REST client

The server client for the Android application was implemented using the Retrofit library and GSON with help the of official documentation. To communicate with the server, you need an interface that defines possible HTTP operations and an instance of the Retrofit.Builder class that will use this interface.

Each interface method must be annotated with @GET, @POST, @PUT, @PATCH, @DELETE, @HEAD or @OPTIONS to understand which REST method should be used for the network call.

The developer can also add an annotation to method parameters:

- @Path – variable substitution for the API endpoint.
- @Query – specifies the query key name with the value of the annotated parameter.
- @Body – payload for the POST call
- @Header – specifies the header with the value of the annotated parameter

The implementation of the required interface can be found below.

```
1  public interface GrammaticaApiInterface {
2      /**
3       * Get all user exercises
4       * @param apiKey user API key
5       */
6      //
7      @GET("exercises")
8      Call<ExercisesResponse> getAllExercises(
           @Header("authorization") String apiKey);
9      /**
10      * Get specific user exercise
11      * @param id exercise id
12      * @param apiKey user API key
13      */
14     //
15     @GET("exercises/{id}")
16     Call<ExercisesResponse> getExercise(@Path("id"
           ) int id, @Header("authorization") String
           apiKey);
17     /**
18      * LogIn user
19      * @param email user email
20      * @param password user password
21      */
22     @POST("login")
23     Call<UserAccount> userLogIn(@Body String email
           , @Body String password);
24     /**
25      * Delete specific user exercise
26      * @param id exercise id
27      * @param apiKey user API key
28      */
29     @DELETE("exercises/{id}")
30     Call<ExercisesResponse> deleteExercise(@Path("
           id") int id, @Header("authorization")
           String apiKey);
31 }
```

Listing 3.7: GrammaticaApiInterface class

As you can see, four endpoints are indicated here - one for user login and three for operations with exercises.

ExercisesResponse and UserAccount are POJO model classes that are used for mapping JSON responses to Java objects.

```
1       @SerializedName("id")
2       private int id;
3       @SerializedName("type")
4       private String type;
5       @SerializedName("exercise")
6       private ArrayList<BaseExercise> exercises;
7       @SerializedName("status")
8       private String status;
9       @SerializedName("created_at")
10      private Timestamp created_at;
```

Listing 3.8: ExercisesResponse fields

Also for sending requests to the server, the developer must use the builder class

```
1  public class GrammaticaApiClient {
2
3      public static final String BASE_URL = "http://
           localhost/grammatica/v1/";
4      private static Retrofit retrofit = null;
5
6      public static Retrofit getClient() {
7          if (retrofit == null) {
8              retrofit = new Retrofit.Builder()
9                      .baseUrl(BASE_URL)
10                     .addConverterFactory(
                           GsonConverterFactory.create
                           ())
11                     .build();
12         }
13         return retrofit;
14     }
15 }
```

Listing 3.9: User login response

Now we can send the request asynchronously using the method `call.enqueue(Callback<T> callback)`.

3.1.6 Used graphics

The color scheme of the application was chosen following the Material Design Guidelines [15] Several vector icons were used under the CC 3.0 BY license. Therefore, the authors are mentioned in the About section of the Settings screen.

3.2 Server-side implementation

The server part is a PHP application that provides a RESTful JSON API. Moreover, as mentioned above, the main goal of the work is to develop an Android application, and therefore, this chapter describes only the most essential aspects of the implementation of the server side.

3.2.1 Slim framework

This framework has been used to simplify the development of a REST framework. Slim is light and easy to understand the framework and does not require much time to master. It supports all the necessary HTTP methods for REST API: GET, POST, PUT and DELETE. Slim can also serve as a middleware layer, which allows the developer to edit and filter HTTP requests and responses. In our case, this feature will be implemented to restrict access to the API using the API key.

3.2.2 Database

MySQL was used to implement the server database. The database model diagram is shown on Figure 3.2

- `users`: All users data goes here
- `exercises`: User exercises are stored in this table.
  Field exercises contains exercise type ("wh", "fi" or "ma"), exersise and status (represents if the exercise done or not). Detailed exercises structure is shown in Listing 2.1.
- `user_exercises`: This table is storing information about users and their exercises.
- `user_stats`: Table used to store user statistics.

3.2.3 Encryption

To ensure the security of user data, it is not advised to store their passwords as plain text on the server. Therefore, in the server application, passwords are encrypted before they are placed to the database. During registration, the password is hashed and placed in the database. When the same user is logging in, the input password is also hashed and then compared to the hash from the database. For hashing, the Blowfish algorithm is used.

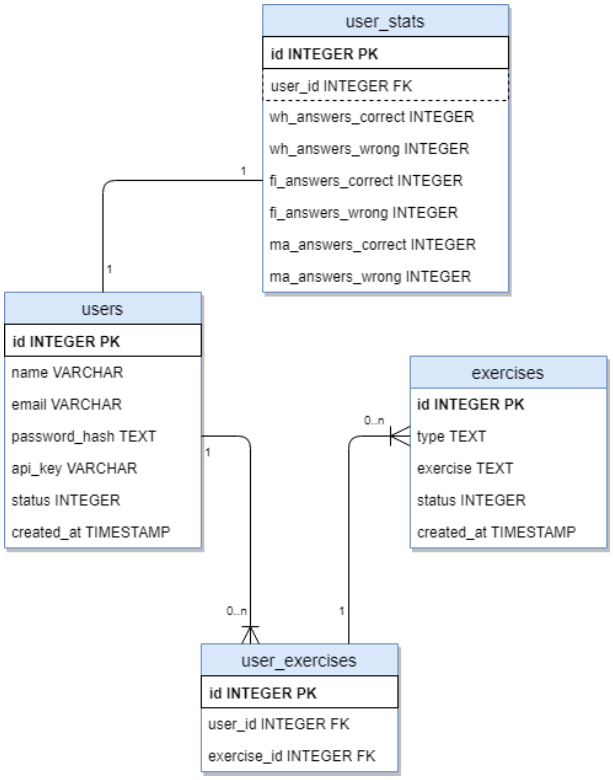

Figure 3.2: Server Database Model

```php
class PassHash {
    // blowfish
    private static $algo = '$2a';
    // cost parameter
    private static $cost = '$10';
    // mainly for internal use
    public static function unique_salt() {
        return substr(sha1(mt_rand()), 0, 22);
    }
    // this will be used to generate a hash
    public static function hash($password) {
        return crypt($password, self::$algo .
                self::$cost .
                '$' . self::unique_salt());
    }
    // this will be used to compare a password
       against a hash
    public static function check_password($hash,
       $password) {
        $full_salt = substr($hash, 0, 29);
        $new_hash = crypt($password, $full_salt);
        return ($hash == $new_hash);
    }
}
```

Listing 3.10: User login response

3.3 Deployment

Diagram of the deployed application is shown on Figure 2.8.

Apache, PHP, and MySQL must be installed on the server. To create a database, the developer must execute the provided SQL query (SQLQuery.sql). This can be done for example in the phpMyAdmin panel.

Because Grammatica has not yet been published to Google Play, to install it, it is needed to make a small setup of the device. Go to Settings, then Security and check Unknown sources. This will allow installing third-party applications on this device.

The application can be built using the gradle console command or Android Studio IDE. As a result, an APK file will be received, which should be copied to the Android device and installed.

User must provide INTERNET permission for the application to work. Internet is needed for communication with the server and MyScript iInk SDK.

Chapter 4

Testing

This chapter describes the method, process and test results of the application. Test results will be analysed, and the application will be corrected.

4.1 Testing by Developer

The purpose of this testing is to ensure the correct operation of the application and search for critical errors. Testing by the developer was carried out every time after the implementation of the new functionality. At the end of development, the application was tested on several Android devices. Devices with different versions of the Android operating system were deliberately selected to make sure that the application works correctly on all supported versions. These devices can be found in the Table 4.1.

	OS version	Display size	Display resolution	RAM
Xiaomi Mi 5s	8.0	5.15	1080 x 1920	3 Gb
Samsung Galaxy Note	6.0.1	10.1	1280 x 800	2 Gb
Samsung Galaxy Tab S4	8.1	10.5	2560 x 1600	4 Gb
Emulator	5.0	7	1200 x 1920	2.5 Gb

Table 4.1: Android devices distribution

4.2 Usability Testing

Usability testing is one of the best ways to test the resulting application for functionality and intuitiveness from the point of view of the average user. There are two types of usability testing: qualitative and quantitative.

In qualitative testing, tests are aimed at analyzing a small number of people representing the target user group of this application. The purpose of such tests is to identify problems with product usability and assess the degree of satisfaction of users of the application. Because Applications are developed with targeting a specific target group, testing with responders from such group provides valuable feedback and potentially valuable tips for improving the quality of the application.

In quantitative testing, tests are aimed at analyzing a large number of users who are not necessarily representatives of the target group. For example, one of the varieties of such tests is the A / B method. Different groups of users show the same element, but with some variations. It can be a small change like a different button name and a relatively large one like a different application flow. Another method of quantitative testing is to connect various analytical services to the product and collect information about users' actions (Google Analytics is an excellent example of such a service). This type of test has its advantages, but the problem is that it targets a large number of people and usually takes place in the form of a beta release for a limited number of users. This means that the application should be in a stage that is close to release, and therefore such testing is not applicable in this thesis

Accordingly, our choice falls on qualitative testing. From the information obtained in, this type of testing consists of the following stages [11]:

- Searching for suitable candidates for testing
- Preparation of the entry survey
- Preparation of test scripts and tasks
- Preparation of the final survey
- Testing
- Results analysis

4.2.1 Preparation and testing

There is no point in taking more than five respondents for this testing method. [16]

Moreover, it is crucial that responders are chosen from the target user group. Therefore, for testing Grammatica, I tried to find only people who are interested in languages, i.e., know at least two languages at a more or less

good level. Five people took part in the final testing, four of whom know more than three languages, and all use Android. Before testing, the respondents completed the initial surveys, with the help of which their brief description was compiled. The survey forms can be found in the Appendix C.

- Respondent 1
  A man, 23-27 years old, is learning a language for study and traveling. Spends 4-8 hours a week learning a language, 1-2 of which on mobile apps. Knows three languages.

- Respondent 2
  A man of 18-23 years old is learning a language to communicate with foreigners. Spends 1-3 hours a week learning a language, of which 3-4 spend on mobile apps. Knows two languages.

- Respondent 3
  Woman, 23-27 years old, learning language for travel and interest. Spends 9-15 hours a week learning a language, of which 4+ on mobile apps. Knows four languages.

- Respondent 4
  A man, 18-23 years old, learns languages just out of interest. Spends 4-8 hours per week on learning, less than 1 hour on mobile apps. Knows three languages.

- Respondent 5
  A woman of 37+ years, learning the language of interest and professional activities. Spends 20+ hours per week on learning, does not use mobile apps. Knows five languages.

Testing took place in an ordinary room, with two respondents were conducted via the Internet. Were deployed local server and high-quality video communication with remote respondents was provided.

During the testing, I was close to users but did not help them. I watched them do the exercises and took notes.

4.2.2 Test scenarios

Here are the scenarios that the respondents had to perform.

1. Complete the Whole Sentence Quiz.
   Go to the Quizzes page, choose Whole Sentence Quiz. After completing all the exercises, check your answers an overview dialog.

2. Complete the Fill-In Quiz.
   Go to the Quizzes page, choose Fill-In Quiz. After completing all the exercises, check your answers in an overview dialog.

3. Complete Matching Quiz.
   Go to the Quizzes page, choose Matching Quiz. After completing all the exercises, check your answers an overview dialog.

4. Check your stats.
   Go to the Profile page and make sure that the statistics correspond with the exercises performed earlier.

5. Clear statistics.
   Go to the Settings page, click Reset your statistics, confirm the action in the dialog. After the message about clearing statistics appears, go to the Profile page and make sure that there is no recorded statistics.

6. Download new exercises.
   Go to the Settings page, click Get new exercises. After the message about the completion of the download appears, go to the Quizzes page, launch any quiz and make sure that new exercises have been added.

Upon completion of all scenarios, each respondent completed a final survey, the purpose of which is to find out the general impression of users about the application. This survey contained both questions addressed to specific parts of the application and general questions about user experience satisfaction.

The final survey can be found in the Appendix C.

4.3 Results analysis

The respondents coped with all the scenarios without any problems. However, during testing, the Internet connection went missing several times. The problem was not on the device or application side, but the provider side. Based on feedback from respondents, several problems were identified:

- **Problem:** In any exercise, after pressing the Answer button, a dialog appears, which allows the user to proceed to the next exercise. If at this time you press the Back button on the device, the dialog will disappear, but the new exercise will not appear.
  Importance: 5/5
  Solution: make dialogues non cancelable.

- **Problem:** when completing any exercise, the user can quickly press the Answer button several times, and the application will record as if the user answered the question several times.
  Importance: 5/5
  Solution: disabled the button until the new exercise is loaded.

- **Problem:** There is no error message on the Settings page when trying to load new exercises if the download failed.
  Importance: 2/5
  Solution: implement toast with an error message.

- **Problem:** It is not clear on the Settings page which buttons are available and which are not, because they are all the same color and they respond to clicking.
  Importance: 1/5
  Solution: disable inaccessible buttons and discolor them.

The most important result of this testing was that the application often incorrectly handles exceptions and errors. I did not anticipate that users will often use the Back button on the device. Moreover, thanks to an unstable Internet connection, it was clear that some error messages are missing.

Conclusion

The purpose of this thesis was to develop a mobile Android application Grammatica for learning foreign languages using handwriting input. This application should support three types of exercises and allow the user to track their progress. It was also necessary to develop a server part for storing user statistics and exercises. Next, it was necessary to test the resulting exercise on real users.

Almost all the goals of this thesis have been fulfilled. Before the start of the design, I analyzed the applications of potential competitors in Section 1.3, which allowed me to understand the main problems of applications of this type and use the best practices in the final Grammatica application.

Then the functional requirements (Section 1.5.1) and use cases (Section 1.6) were determined.

After that, the process of designing the application itself and the server part followed. During the designing part, special attention was paid to the user interface (Section 2.2.1).

Then, based on the design, the server and client parts of the application were implemented (Chapter 3).

In the final part of this thesis, usability testing of the final application was conducted with the help of real users (Chapter 4).

One task is partially not fulfilled - the ability to save user progress in the cloud. On the server side, the necessary API methods for this were implemented, but on the client side, they were not implemented in any way.

The result of this thesis is a mobile application for Android and a server that communicate with each other through a REST service.

And although the application is able to perform its core functions, its development is not yet finished. Here is a list of things that could be implemented in the future:

- Implement storing user statistics on the server

- On the server side, API methods are implemented that allow users to upload their own exercises to the server. However, these methods are not used in the client. It is needed to design and implement an exercise constructor which allows users to create their own exercises inside the Android application.
- After completing the above points, publish the app on the Play Market.
- Then you can pay attention to designing and creating a database with exercises. Possible approaches are described in Section 1.7.

Bibliography

[1] From busuu to Babbel, language-learning startups adapt to thrive. *The Guardian [online]*, March 2017, [cit. 2019-04-20]. Available from: https://www.theguardian.com/small-business-network/2017/mar/07/busuu-babble-language-learning-startups-adapt-thrive

[2] Strategy Analytics. https://www.strategyanalytics.com/, 2018, [Online; accessed 19-04-01].

[3] Leitner, S. *How to learn to learn*. Freiburg i. Br., 2003, ISBN 3-451-05060-9.

[4] T-Mobile has sold 1 million G1 Android phones. *TOM KRAZIT [online]*, April 2009, [cit. 2019-04-20]. Available from: https://www.cnet.com/news/t-mobile-has-sold-1-million-g1-android-phones/

[5] Statista. https://www.statista.com/, 2018, [Online; accessed 19-03-01].

[6] How to use the Play Console. https://support.google.com/googleplay/android-developer/answer/6112435, [Online; accessed 18-12-15].

[7] Apple Developer Program. https://developer.apple.com/programs/how-it-works/, [Online; accessed 18-12-15].

[8] Distribution dashboard. https://developer.android.com/about/dashboards/, 2019, [Online; accessed 19-03-05].

[9] Ivar Jacobson, K. B., Ian Spence. *USE-CASE 2.0, The Guide to Succeeding with Use Cases*. 2011, ISBN 978-0201544350.

[10] Galitz, W. O. *The essential guide to user interface design: an introduction to GUI design principles and techniques*. Wiley Pub, third edition, ISBN 978-0-470-05342-3.

[11] Žizkovský, P. *User Interface Design*. Unpublished lectures, 2017.

[12] 10 Usability Heuristics for User Interface Designs. *Nielsen Norman Group [online]*, April 1994, [cit. 2019-04-20]. Available from: https://www.nngroup.com/articles/ten-usability-heuristics/

[13] Guide to app architecture. https://developer.android.com/jetpack/docs/guide/#fetch-data, [Online; accessed 19-03-07].

[14] Cross-platform handwriting recognition and Interactive Ink APIs. https://dev.myscript.com, [Online; accessed 18-12-15].

[15] Material.IO. *The color system*. 2019. Available from: https://material.io/design/color/the-color-system.html

[16] Why You Only Need to Test with 5 Users. *Nielsen Norman Group [online]*, March 2000, [cit. 2019-04-20]. Available from: https://www.nngroup.com/articles/why-you-only-need-to-test-with-5-users/

Appendix A

Acronyms

API Application Programming Interface

HTTP Hypertext Transfer Protocol

JSON JavaScript Object Notation

OS Operating System

REST Representational State Transfer

SDK Software Development Kit

SQL Structured Query Language

UI User Interface

URL Uniform Resource Locator

XML Extensible Markup Language

Appendix B

Screenshots of the final application

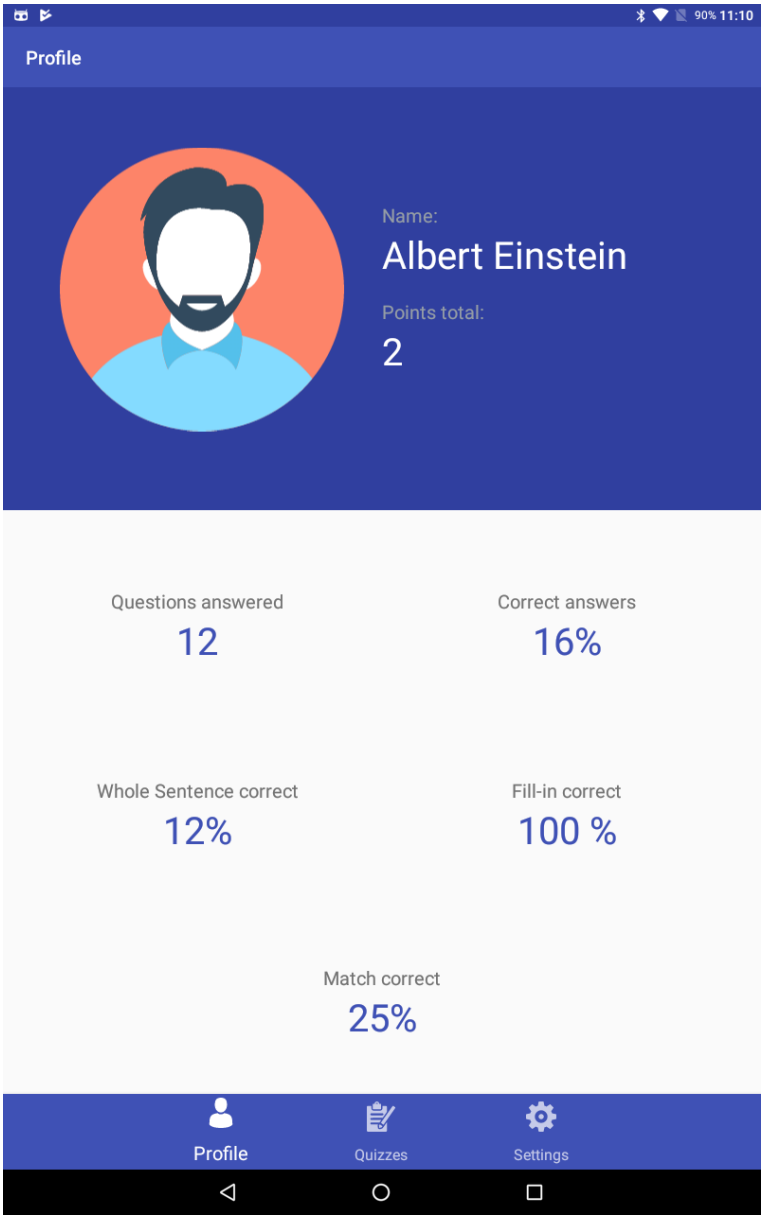

Figure B.1: Profile screen

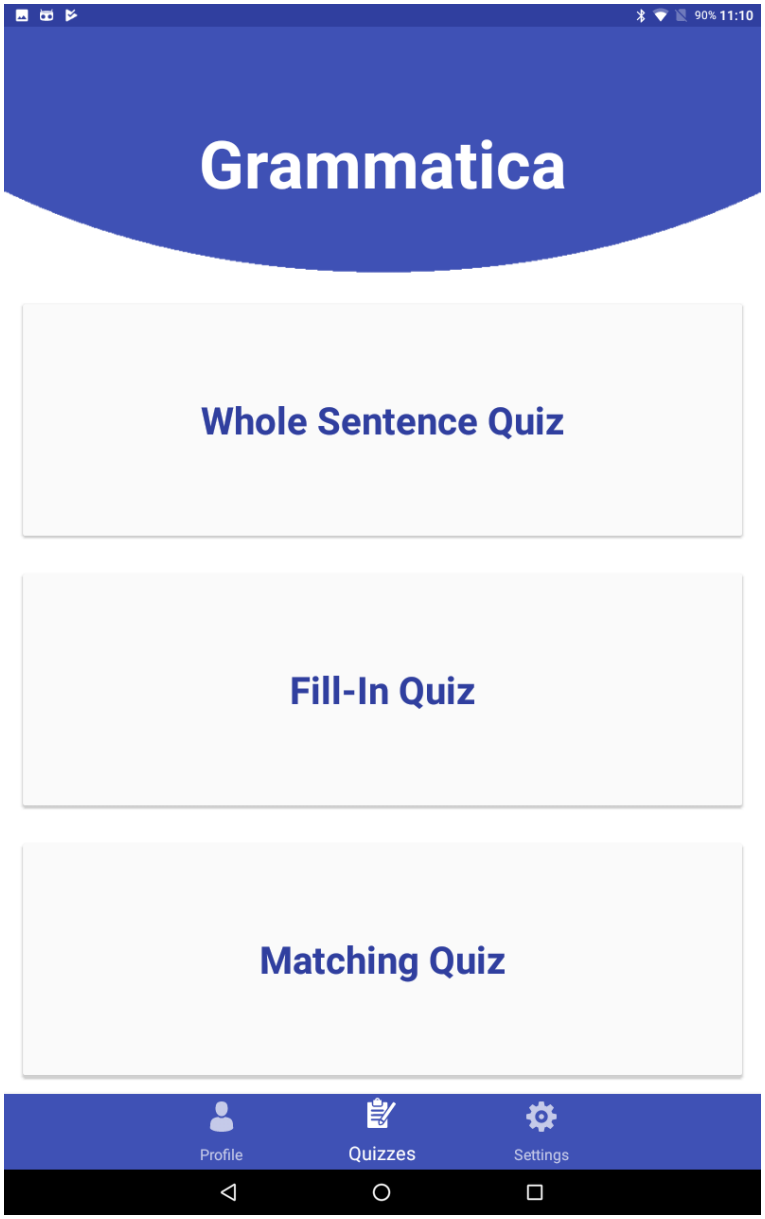

Figure B.2: Quizzes screen

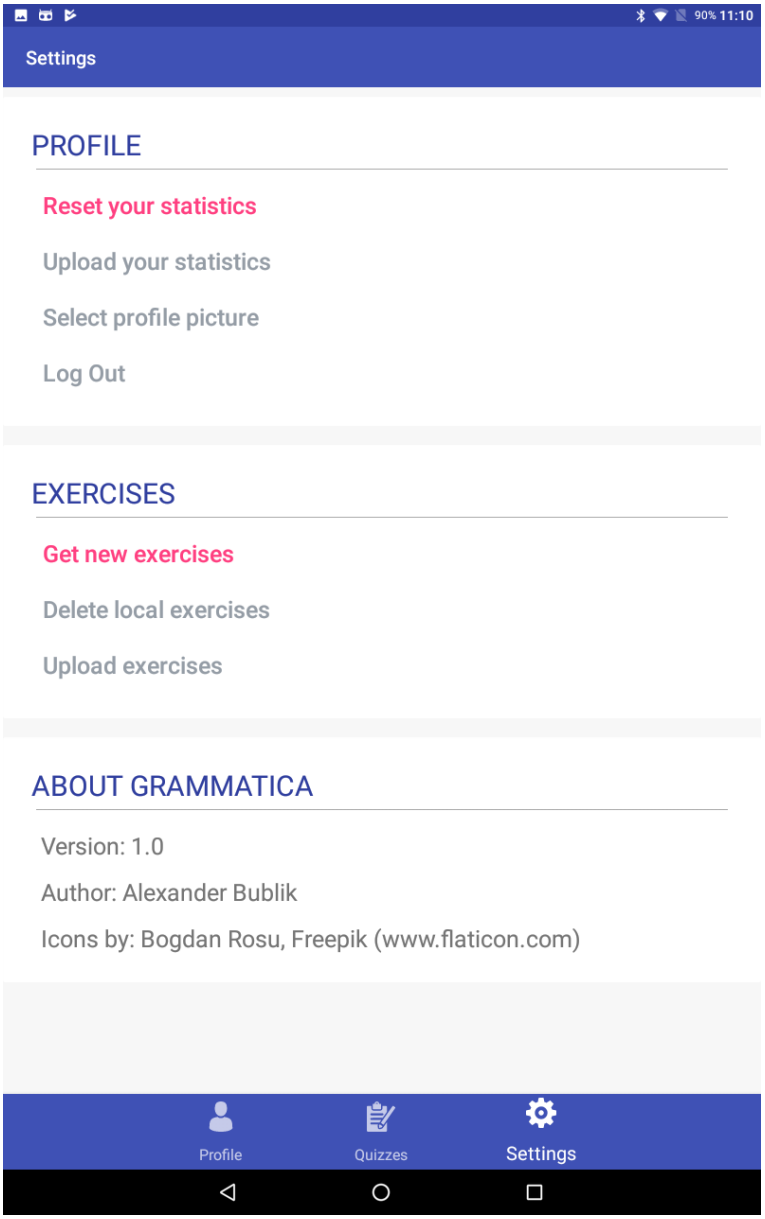

Figure B.3: Settings screen

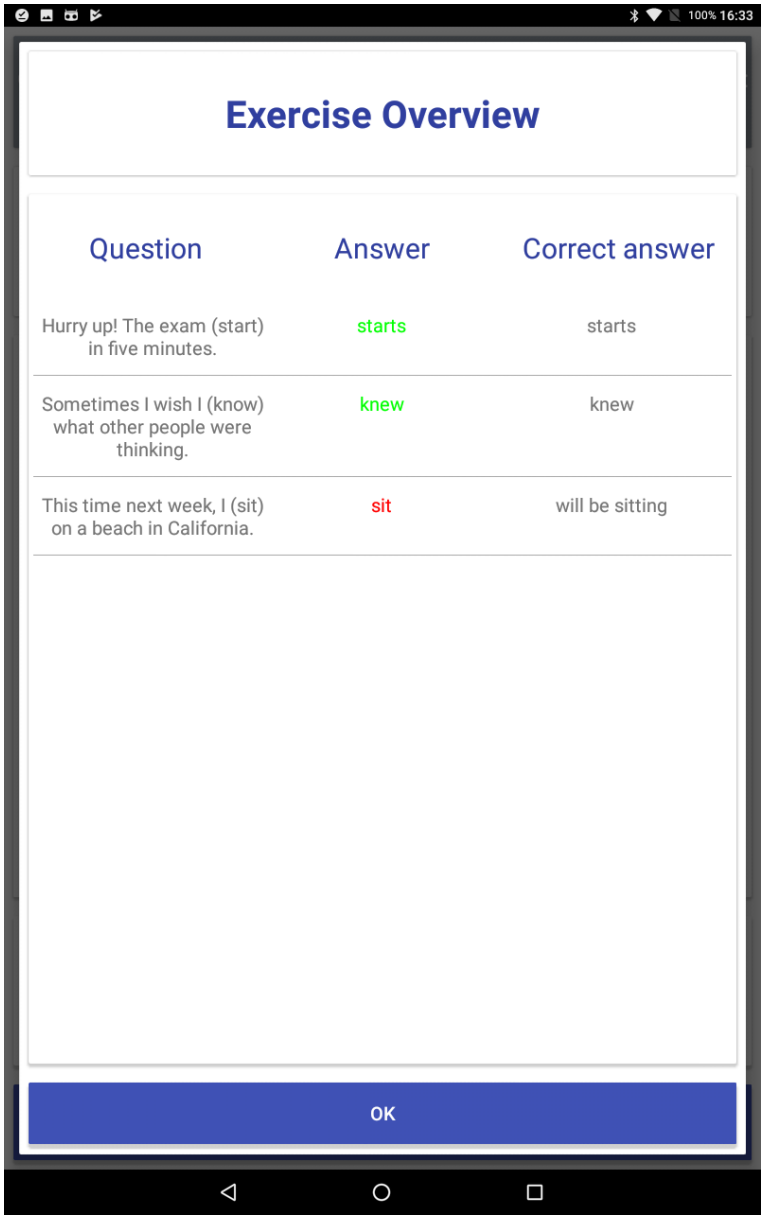

Figure B.4: Exercise overview

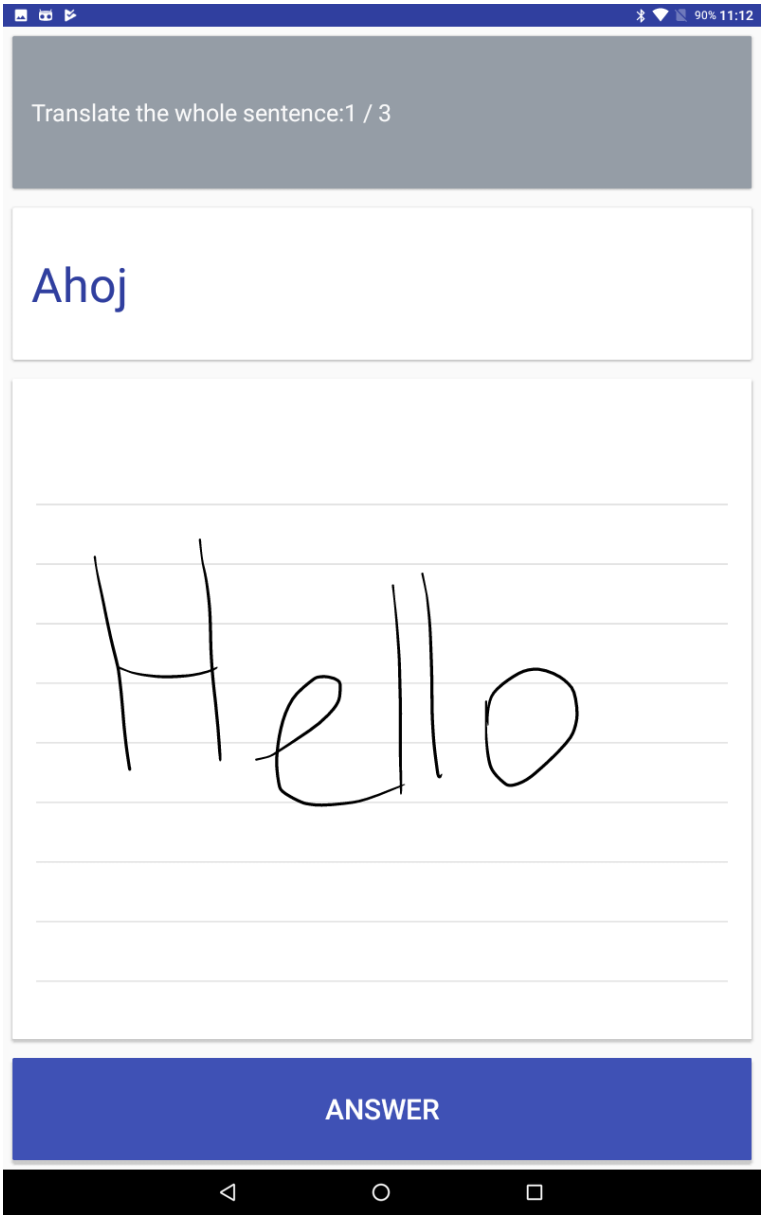

Figure B.5: Whole Sentence exercise

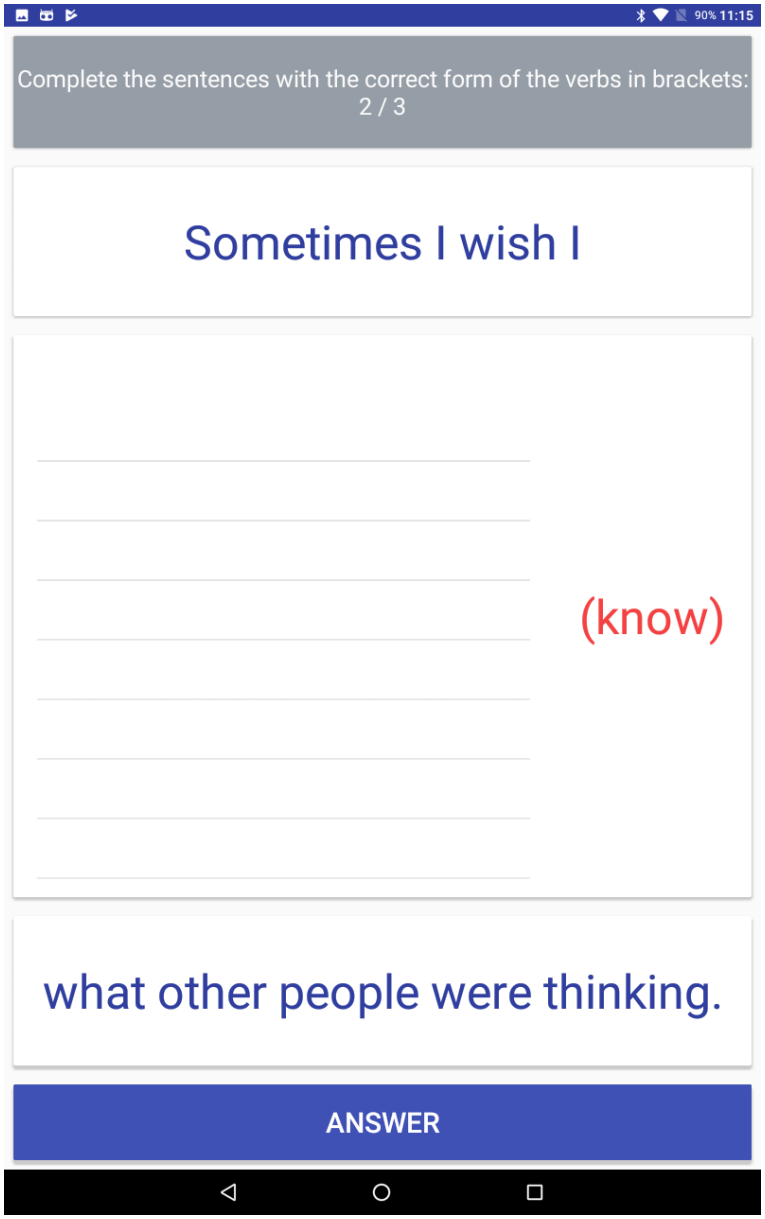

Figure B.6: Fill-in exercise

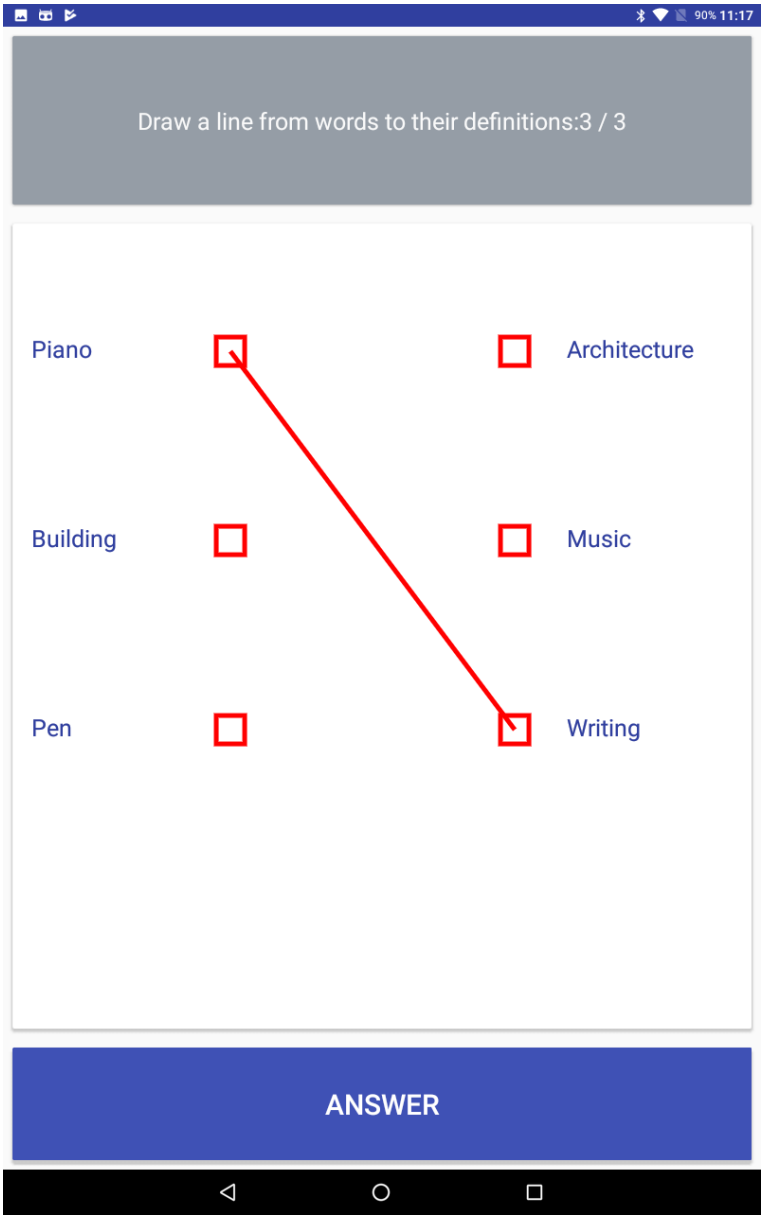

Figure B.7: Matching exercise

Appendix C

Survey forms

C.1 Entry survey

1. How old are you?
    a) 18-23 [Respondent 2, Respondent 4]
    b) 23-27 [Respondent 1, Respondent 3]
    c) 28-36
    d) 37+ [Respondent 5]
2. How many languages do you know?
    a) 2 [Respondent 2]
    b) 3 [Respondent 1, Respondent 4]
    c) 4 [Respondent 3]
    d) 4+ [Respondent 5]
3. Why do you learn new languages?
    a) study [Respondent 1]
    b) travel [Respondent 3]
    c) communication with foreign contacts [Respondent 2]
    d) professional activity [Respondent 5]
    e) just out of interest [Respondent 3, Respondent 4, Respondent 5]
4. How many hours per week do you spend learning languages?
    a) 1-3 [Respondent 2]
    b) 4-8 [Respondent 1, Respondent 4]
    c) 9-15 [Respondent 3]

d) 20+ [Respondent 5]

5. If you use mobile apps to learn languages, how many hours do you spend on them per week?

   a) Not using [Respondent 5]
   b) less than 1 [Respondent 4]
   c) 1-2 [Respondent 1]
   d) 3-4 [Respondent 2]
   e) 4+ [Respondent 3]

6. How well are you familiar with the Android operating system?

   a) Do not use at all
   b) Sometimes I use device for calls [Respondent 5]
   c) I often use several applications [Respondent 1, Respondent 3, Respondent 4]
   d) I constantly use many applications and I know most of the functions of my device. [Respondent 2]

C.2 Final survey

1. Was it immediately obvious where the quizzes are and how to start them?

   a) Definitely yes [Respondent 1, Respondent 2 ,Respondent 3, Respondent 4]
   b) Rather yes than no
   c) I do not know [Respondent 5]
   d) Rather no than yes
   e) Definitely no

2. Did you immediately recognize the essence of different types of exercises from their titles?

   a) Definitely yes [Respondent 5]
   b) Rather yes than no [Respondent 1, Respondent 3 ,Respondent 4]
   c) I do not know [Respondent 2]
   d) Rather no than yes
   e) Definitely no

3. Did you find it useful to collect statistics on your answers?

a) Definitely yes [Respondent 1, Respondent 2 ,Respondent 4]
b) Rather yes than no
c) I do not know [Respondent 5]
d) Rather no than yes [Respondent 3]
e) Definitely no

4. Was it hard for you to get used to handwriting input?

a) Definitely yes [Respondent 2]
b) Rather yes than no [Respondent 5]
c) I do not know [Respondent 1, Respondent 3 ,Respondent 4]
d) Rather no than yes
e) Definitely no

5. Did you like the app? (rate from 1 to 5)

a) 5 [Respondent 1]
b) 4 [Respondent 3, Respondent 5]
c) 3 [Respondent 2, Respondent 4]

6. How much would you rate your overall experience with the application? (rate from 1 to 5)

a) 4 [Respondent 1, Respondent 2, Respondent 5]
b) 3 [Respondent 3, Respondent 4]

7. What features do you think the application lacks?

a) Respondent 1: A quizzes consisting of mixed types of exercises.
b) Respondent 3: Mixed quiz type and exercise distribution by difficulty level.
c) Respondent 4: Any social functions. For example, comparing your results with friends.