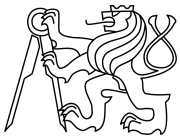
CZECH TECHNICAL UNIVERSITY IN PRAGUE
FACULTY OF INFORMATION TECHNOLOGY



ASSIGNMENT OF MASTER'S THESIS

Title:	Mobile app for Hrave.cz
Student:	Bc. Ondřej Paška
Supervisor:	Ing. Ondřej Menčl
Study Programme:	Informatics
Study Branch:	Web and Software Engineering
Department:	Department of Software Engineering
Validity:	Until the end of summer semester 2016/17

Instructions

The web portal Hrave.cz, which belongs to the Educasoft company, aims to prepare high-school students for their state final exams and to prepare them for entrance exams for universities. It includes lectures and many different types of exercises and includes some gamification machanisms. The task is to create a mobile application that replicates most of the functions of the portal. The app will communicate with the existing API server. It is expected to include user login and registration, user progress visualization, lectures, and all types of exercises. The app should be modular and ready to be easily maintained and extended. Design and graphics assets for the UI will be provided by the graphics department of the Educasoft company.

1. Analyze the web application and specify functional and non-functional requirements for a mobile application.
2. Design the mobile application.
3. Choose an implementation platform.
4. Implement the application, test it, and document it.

References

Will be provided by the supervisor.

L.S.

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

prof. Ing. Pavel Tvrdík, CSc.
Dean

Prague January 24, 2016

Czech Technical University in Prague
Faculty of Information Technology
Department of theoretical computer science

Master's thesis

Mobile app for Hrave.cz

Bc. Ondřej Paška

Supervisor: Ing. Ondřej Menčl

9th May 2016

Acknowledgements

I would like to thank my family for their support during my studies. Special thanks to Tereza Novická for proofreading and correcting this text.

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 9th May 2016 .....

Czech Technical University in Prague
Faculty of Information Technology

Citation of this thesis

Paška, Ondřej. *Mobile app for Hrave.cz.* Master's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2016.

Abstrakt

Předmětem této diplomové práce je vytvoření mobilní aplikace k existující e-learningové platformě Hrave.cz. Práce obsahuje popis této služby a rozbor podobných řešení na mobilnách platformách. Jsou probrány možné cesty mobilního vývoje. Aplikace je implementována pomocí multiplatformního frameworku Adobe AIR pro Android a iOS, otestována a vydána.

Klíčová slova LMS, učení, maturita, e-learning, mobilni aplikace, Android, multiplatformní

Abstract

The goal of this thesis is to create a mobile application for the existing Hrave.cz e-learning platform. The service is analyzed and similar solutions on mobile platforms are explored. Possible ways of implementing the mobile service are discussed. The application is implemented in the Adobe AIR framework for Android and iOS, tested, and deployed.

Keywords LMS, e-learning, students, Android, mobile, multiplatform

Contents

List of Figures

List of Tables

Introduction

Motivation and objectives

Utilizing information technology in education is not a new concept. The idea of electronic learning (or e-learning) is thought by many to be a key to effectively spreading knowledge in the 21st century[1]. E-learning commonly complements traditional learning at Czech universities and high schools. In 2010 a standardised test at the end of secondary school (the so-called state maturita) was introduced to Czech schools[2] and more secondary schools and universities started using standardised entrance exams called Scio[3]. There were suddenly two standard exams the majority of students needed to prepared for.

Hrave is a project aimed at helping students preparing for these exams in a fun and motivational way and has attracted a lot of users since it was launched in 2014. The service is accessed through a web browser and was designed with desktop computers in mind. The trend, however, is that more and more people in the Czech Republic access the web through their smartphones and tablets[4]. Furthermore, people prefer to access services through mobile apps, rather than mobile web sites.[5] Althought mobile e-learning systems are common in the US (see section 1.3), there is relatively small competition in the Czech mobile e-learning market. With this in mind, I agreed to work with Educasoft company to tap into this market, by creating a mobile app that enables students to use Hrave e-learning platform on their mobile phones and tablets. In addition to this, Hrave software can be used as a platform for any kind of learning, which forms the core of Educasoft B2B operations. The mobile app should also work within these other domains.

In this thesis I will first describe the existing Hrave platform and the way content is created and structured and explore some existing solutions on the mobile market. Then I will present an analysis of the new mobile system and discuss the design. Thirdly I will show how I implemented the application and finally how it was tested and deployed.

Chapter 1

Analysis

1.1 Educasoft s.r.o.

The company was created in 2014 with the intention of using modern technologies in education. Their first project was Hrave.cz, but today they also extend and support the learning management system (LMS) iTrivio (itrivio.cz) for managing know-how, learning, and training in schools and businesses.

1.2 Hrave.cz

Hrave is a Czech online learning tool for students. It now has modules for the final high school exam (maturita), university entrance exams (NSZ - OSP) and entrance exams for eight-year Gymnasiums (gymnázia). It features:

- Professionally prepared lectures
- Many types of exercises
- Gamification mechanics - experience points (XPs), earning diamonds, practice arena
- Leaderboards for competing with friends

Some content is provided for free, but to unlock all the lessons, users must purchase a license in the eShop section of the web. Licenses are usually valid for one or two years. Registration is required and is possible through email or with one-click Facebook login.

1.2.1 Architecture

Existing system consists of a server application (backend) and a web application (frontend). Backend is written in Java using Hibernate and Spring

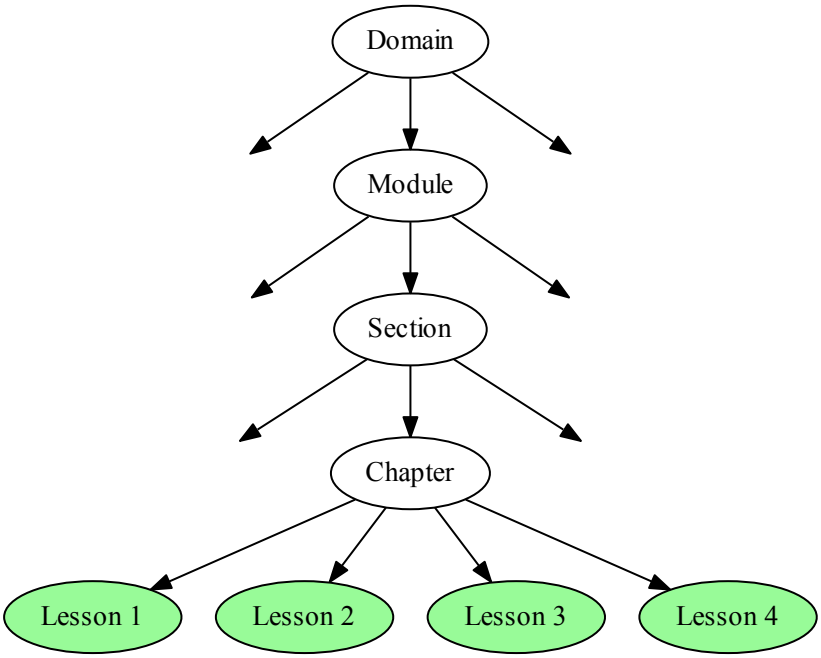


Figure 1.1: Part of Hrave lesson tree

frameworks and frontend uses HTML, CSS and Javascript with JQuery and Angular frameworks. The communication between frontend and backend happens throught a HTTP API using JSON data format. Hrave itself is a generic e-learning platform, which can be licensed for other uses than student preparation. Hrave.cz is just one domain where it is used.

1.2.2 Lesson Structure

Every module is divided into a set of small self-contained lessons. Lessons are categorized in a tree structure. The root has children corresponding to the modules (Maturita, NSZ-OSP, ...) and those can have arbitrary levels with the leafs being the lessons themselves. Figure 1.1 shows a part of this tree. Each lesson is composed of lectures explaining the topic and exercises (questions).

1.2.3 Editor

Lesson content is prepared by teachers or experts in a WYSIWYG HTML editor. Hrave uses TinyMCE for this, which is an open-source, full-featured, yet simple to use editor[6]. It allows basic formatting with some predefined styles, raster and vector images, math notation and equations and sounds. Youtube videos can be embedded as well.

Users go through a lesson one part at a time and their progress is tracked and displayed, so they know how well they are doing. Some modules include a final test, which simulates a real-life exam. This case has a special user interface (UI) where the user can see all the questions at once and has a time limit. Only after the test is finished can the student see their evaluation and review correct answers.

1.2.4 Question Types

Hrave currently supports five different types of questions. Example questions can be seen in figure 1.2.

Simple question is an open question, where user can submit any text input. Maximum length can be specified. Evaluation contains a list of possible answers.

Choice question contains several possibilities that can be selected. Any number of options can be selected.

Connect question contains several parts that must be matched together. Left and right parts and their labels are specified.

True/False question contains several statements that the user evaluates as true or false. Specific labels for the true and false options can be specified.

TextFill question contains a text with one or more words that must be chosen from a list.

1.2.5 Gamification

Keeping users engaged is one of the challenges of any e-learning system. Gamification is at the center of many popular e-learning platforms like Codeacademy and Duolingo[7]. It aims to utilize the way games keep players motivated and engaged in non-game areas and it has been a big trend in the last couple of years in both industry and academic research[8].

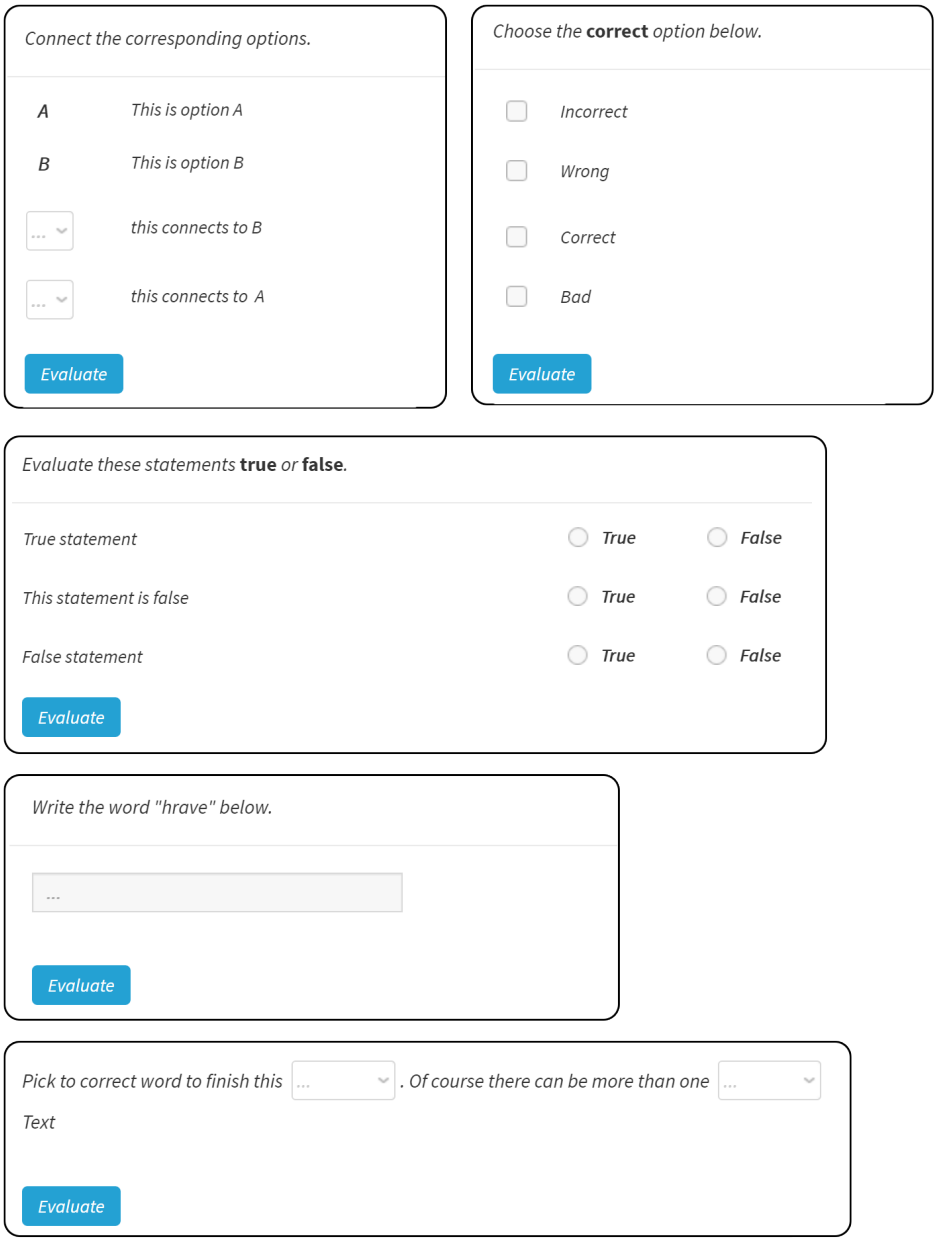


Figure 1.2: Hrave Question Types

Typical gamification mechanics include:

- Points
- Leaderboards
- Achievements
- Badges
- Levels
- Story/Theme
- Clear goals [8]

Research shows that in a given system, gamification increases student participation significantly[9]. There are several ways gamification accomplishes this. Firstly, it targets the users' emotions by giving them a positive reward for every task completed, which gives them a sense of achievement. Secondly, it uses social interactions (e.g. with leaderboards) to utilize a natural need for status and recognition.[9]

1.2.5.1 Gamification Elements of Hrave.cz

In Hrave, gamification is realized by giving users XPs after each lesson, based on how well they did. Based on the percentage of correct answers the users are also awarded a diamond for each completed lesson. A silver diamond is awarded for getting the minimal required points for the lesson and a gold diamond is awarded for having everything correct. Diamonds are also displayed for whole chapters of lessons, up through the lesson tree (see figure 1.1), where they reflect the average achieved diamonds in child branches. For example: if a user completes half the lessons in a chapter (gets a gold diamond for them) he gets half a gold diamond for the whole chapter.

Based on their XPs, the users gain 'levels' in each module. The aim is to motivate the student to gain XPs.

Another gamification technique is to include a leaderboard for friends. The user can connect with friends via email or Facebook and compare their XPs in each subject (module). This creates healthy competition and additional motivation for the user.

1.2.5.2 Arena

Lessons for the maturita exam can be practiced in a special game mode called Arena. Here, the student fights against the personified 'Evil Maturita'. Questions are shown and when the user answers correctly, his opponent gets 'hit'.

User loses hitpoints (health) by answering incorrectly. The user can skip questions, but does not get to see the correct answers until after a round is over. Future plans include player-against-player competition. Users receive special purple diamonds for playing in Arena.

1.3 Similar Existing Solutions

The aim of this thesis is to bring some of this functionality to the mobile platform. Firstly, let us examine some similar existing solutions. There are a few mobile applications on AppStore and Google Play that aim to help students studying for maturita exams, however, most of them suffer from very poor user ratings. Overall, the competition for the (admittedly small) market of mobile learning in the Czech Republic seems to be almost non-existent.

1.3.1 Maturita SK

One exception is the Slovakian Maturita SK based on a student website zones.sk, which is well-liked by its users and has 50 000 - 100 000 downloads on Google Play. It offers free study materials from all subjects, but it has no exercises or gamification mechanics. The quality of the materials varies substantially. It also suffers from some UI problems - the back arrow does not work, the text is often weirdly formatted, and some math equations are not displayed correctly. Some lessons are redirected to Google Drive files.

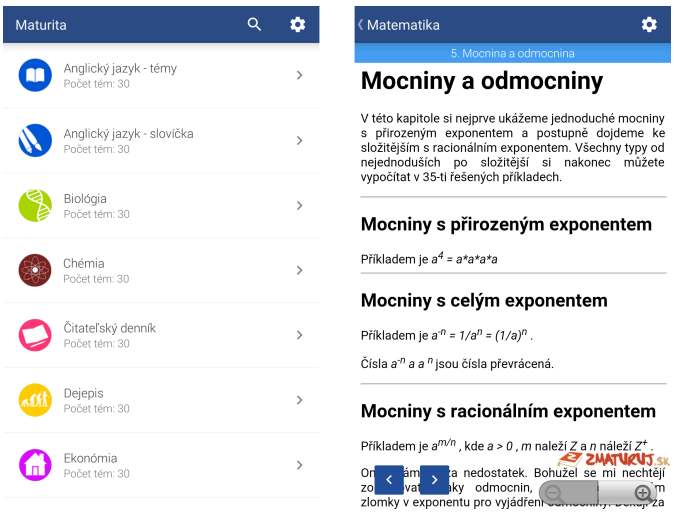


Figure 1.3: Maturita SK on Android

1.3.2 Maturita Quiz

This app is among the first results for the search term 'maturita' on both App Store and Google Play, therefore it should be a direct competitor to Hrave mobile application. It has support for subjects - Math, Czech, English, and German. It offers only multiple choice questions with some supplementary explanations. Some of the topics are for free, for others you have to buy some in-app currency to unlock them. It also contains bottom-page advertisements.

The UI does not conform to the standards on either platform and is quite unintuitive and cumbersome. For example, when selecting a subject, the user must first tap the subject and then click continue, which is quite unnecessary. Also, when the explanation for a question is displayed, the correct answer can no longer be seen.

Figure 1.4: Maturita Quiz on Android

1.3.3 Duolingo

With millions of downloads and user ratings above 4.6 stars in both App Store and Google Play, this application is among the most successful in the area of mobile learning. It is focused on languages, but is very similar to what Hrave mobile should be, mainly it features: a lesson structure with topics and lessons, multiple types of questions, gamification, and friends. One thing that it lacks is the ability to save lessons for offline use. As of April 2016, the only course available for Czech speakers is English. All of their content is available for free without ads. The company's business model is based on selling translations from high-level users[7]. The UI is an example of an uncluttered, well-arranged, and user-friendly design.

Application screens (screenshots taken from Android version 3.20.1):

- Landing screen, the user continues to sign-in or there's onboarding for new users. 1.5a
- Sign-in screen 1.5b
- Topic screen, after initial sign-in, this becomes the default screen. Some gamification notifications can be seen. 1.5c
- Topic introduction screen 1.5d

1.3.4 Khan Academy

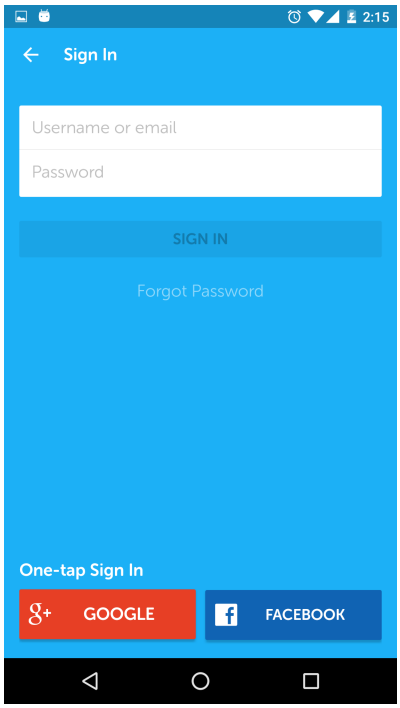
The last mobile application I would like to mention is Khan Academy, a popular video-based learning service available on the web, Android and iOS. It includes dozens of subjects with many subcategories. Each topic is covered in a series of 8-15 minute videos with practical exercises in between (these seem to be missing on mobile devices) and students are motivated by gamification elements 1.6d. All content is for free. It is in English, however some videos have Czech subtitles. What is interesting is the possibility to download lessons for offline viewing. Users first select a video to be added to their 'list' and later they can download any video for offline viewing 1.6c. The Android application kept displaying the 'You are offline' message, even though the internet connection was active at the time, however online lessons could still be viewed. The UI can seem too cluttered and confusing.

1.4 Choosing a Platform

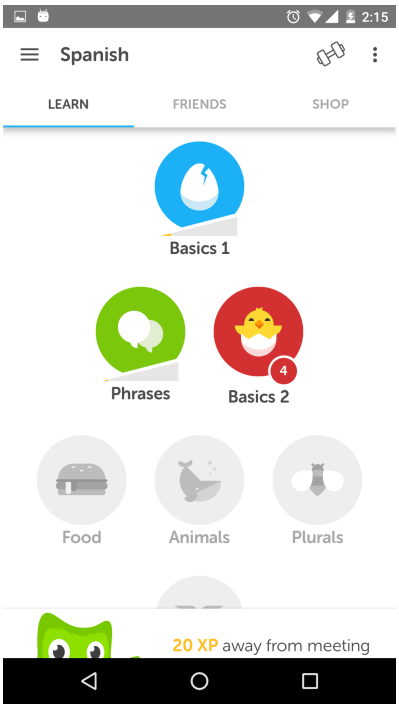
When choosing a platform, the goal is usually to cover most of the users while minimizing the cost. According to IDC[10] worldwide market share of mobile OS is divided as follows: Google's Android dominates with 82.8%, Apple's iOS is at 13.9%, and a distant third place is held by Windows Phone with 2.6%, other options are below one percent and therefore not relevant to this discussion. In the Czech Republic, the trend is similar with Android at 72.5%, iOS at 17.3% and Windows Phone at 5.5% of the market.[11] Android therefore seems like a must for mobile applications, unless we are targeting a very specific demographic. For this thesis, it was decided to include iOS as well in order to capture the most of the market. There are several ways to target users in both ecosystems. The important thing here is that mobile software distribution is more restricted than that on traditional desktop computers.

On Android, the default way to get software is from Google Play Store. It is possible to download and install software from other sources, but it must be explicitly allowed in the settings.

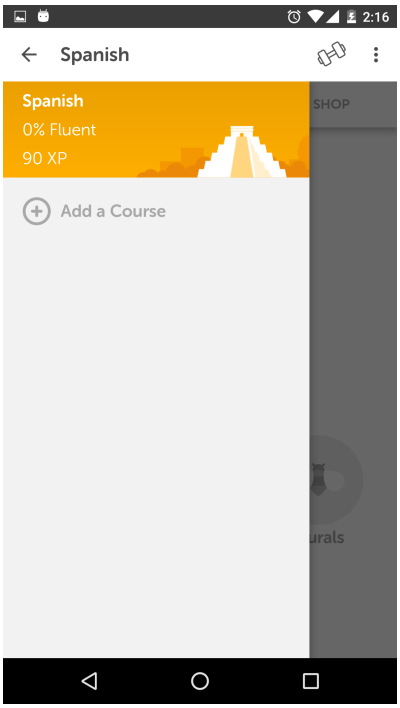
(a) Landing screen



(b) Sign-in screen

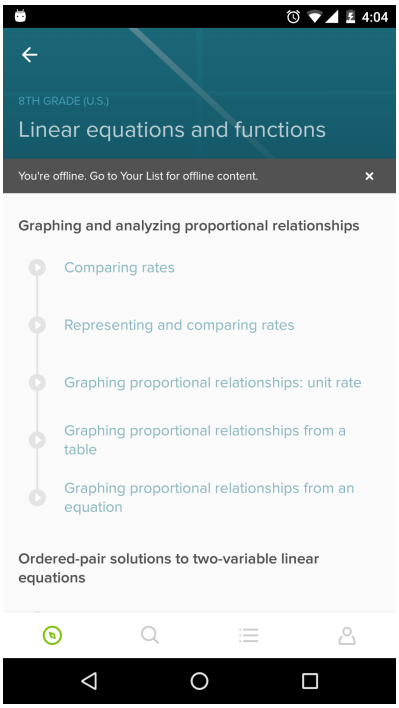


(c) Topic screen

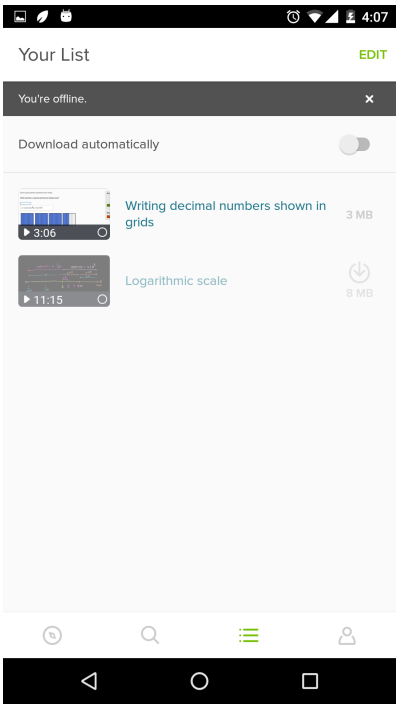


(d) Drawer navigation

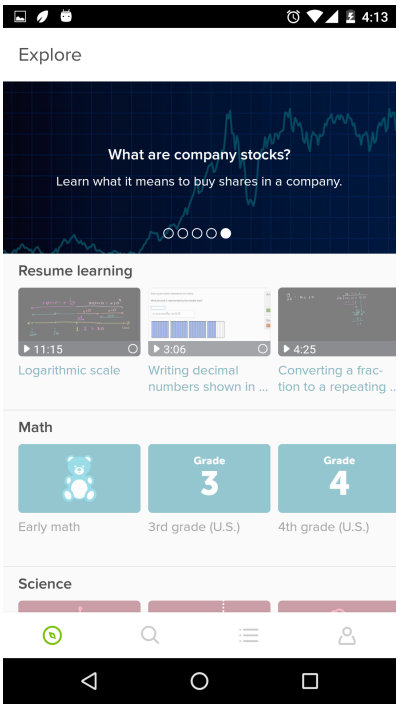
Figure 1.5: Duolingo on Android



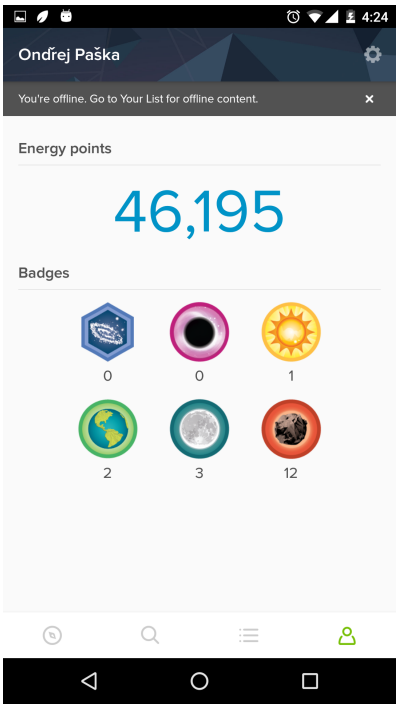
(a) Topic screen



(b) List of offline content



(c) Main screen



(d) Gamification elements

Figure 1.6: Khan Academy on Android

On iOS, the AppStore is the software distribution platform and software from other sources is not permitted.

In the next section, I will discuss several options for multiplatform development of mobile applications.

1.4.1 Web Apps

The easiest way to allow mobile users to access the existing website is through the mobile web browser. This approach, however, has several disadvantages [12].

- The existing website would have to be rewritten to be more responsive.
- The result looks and feels different, and usability becomes an issue.
- Cannot access native components (push notifications, vibrations, etc. . .)
- Works only online

Also, as was already mentioned, users prefer to interact with services throught apps on their mobile devices[5].

1.4.2 Native

The best solution would be to create a native application for each platform. That, however, would be unfeasible for a single developer with the time allocation, and there was no budget for a second mobile developer at the company. Furthermore, native iOS development is only possible on Apple computers[13], which means more cost and inconvenience. Therefore I looked for a crossplatform solution which would enable me to use a single codebase to target both platforms.

1.4.3 Hybrid Apps

Hybrid apps ship as native apps, but under the hood they use a web view to display content made by web technologies like HTML, CSS, and JavaScript. Apache Cordoba is one of the most popular hybrid app frameworks at this moment. It started as PhoneGap, but was opensourced and contributed to the Apache Software Foundation (ASF)[14]. Looking through their showcase applications, they mostly suffered both bad performance and user interface. Furthermore, the build process seems to be greatly convoluted, to the point that Adobe provides a cloud tool for it[15]. Facebook CEO Mark Zuckerberg said that pursuing the hybrid approach on mobile was one of his biggest mistakes[16].

One of the advantages of this approach is that there are a lot of quality UI frameworks specifically made for mobile[17] and most importantly, it would

make it naturally easier to display the lesson content, which is structured in HTML and styled with CSS.

1.4.4 ReactNative

ReactNative, a new opensource framework that emerged recently, is worth mentioning. It is backed by Facebook [18] and it lies somewhere on the border between the hybrid and native approach. It uses Javascript (it is based on React.js) to create and work with native components. With this approach, you still need to write a substantial amount of platform specific code, but can keep a common Javascript code core. The final application looks and behaves exactly like a native app. React Native is still very new (the latest version is V0.22) and under development. OS X is needed for iOS development and Android development on Windows is marked as 'experimental'[18]. For these reasons I decided not to experiment with it just yet, but it is definitely worth following in the future.

1.4.5 Adobe AIR

I will now briefly describe Adobe AIR and how it evolved in recent years and then weigh its advantages and disadvantages.

1.4.5.1 Overview

Adobe AIR is a multiplatform development framework created by Adobe[19]. It is based around Flash technologies, but has native capabilities like file system access, web sockets, local databases, and others. Applications for it are written in ActionScript 3 (AS3) and they behave and run just like native apps on target devices[20]. Its target platforms are Windows, Mac OS, Android, iOS, and BlackBerry Tablet OS[21].

1.4.5.2 ActionScript 3

ActionScript 3 is based on EcmaScript, the international standardized programming language for scripting, which is also the basis for JavaScript, however ActionScript is statically typed, so it is more similar to TypeScript in this respect. It is a very mature language and has features including native XML support, 2D and 3D graphics acceleration, and concurrency on all platforms (via Workers)[22].

1.4.5.3 Native Extensions

AIR Native Extensions (ANEs) give developers the freedom to create certain parts of their program in the native code of their target platform[20]. This way it is possible to have a common code base and still leverage the capabilities

of each platform. Some ANEs are provided by Adobe for free (e.g. vibration and GameCenter), some can be purchased from third party companies, and others can be found open-sourced online.

1.4.5.4 Stage3D

The introduction of Stage3D API in Flash Player 11 enabled developers to use the GPU to accelerate graphics on all platforms[23]. The Starling framework was created to shield the developer from the complexities of Stage3D when creating 2D content. It delivers the performance needed for interactive content on mobile devices and it was officially supported by Adobe[24]. Starling is still actively maintained and new features and performance enhancements are added regularly. It has also a very active and welcoming community online[25].

1.4.5.5 Feathers

Starling is a game engine and was designed and optimised to deliver great performance on mobile. This, coupled with the fact that mobile devices are getting more powerful each year, encouraged developers to develop non-game mobile applications in that framework. Feathers is a UI framework, built on top of Starling, that makes it possible to easily create extensible, skinable components, and supports layouts and themes[26]. In 2015 Feathers came out with Feathers SDK: a toolbox that enables developers to declare Feathers or even custom components in MXML files. They will be described in more detail in section 3.2.

To summarize the advantages of Adobe AIR:

- Delivers consistent results on all platforms with code crosscompilation for iOS.
- Is a mature technology with many tools and libraries available.
- Can access platform specific code with ANEs.
- Makes it possible to develop iOS applications on Windows.
- Free for commercial development

The disadvantages of creating apps in Adobe AIR include:

- The result will not feel truly native.
- Some features might only be accessed in a round-about way through an ANE.
- Unlike hybrid-apps, it only supports a small subset of HTML and CSS.
- The runtime adds additional 9MB to the resulting package.

1.4.6 Conclusion

In the end I decided to implement the applications for both platforms (Android and iOS) with Adobe AIR. I must admit this is partly because of my positive experience with it in my previous projects. Looking at the competing approaches, none of their specifications convinced me to switch from Adobe AIR, with the exception of native development, which I would recommend to the company if they had more resources.

Further information about Adobe AIR can be also found in my bachelor thesis [27].

1.5 Requirements

Hrave mobile application is intended as an addition to the website, its scope was limited so that it would be manageable by a single developer in the course of several months. Specifically, the following parts are **not** going to be part of the design and implementation, but they could be added in the future:

- eShop - users will only have the licenses they bought through the website
- Arena
- Administration or content management, users will not create or edit any content
- Final exam preparation
- Facebook integration (will be added in a later version)

1.5.1 Functional Requirements

With the scope in mind we can formulate the following functional requirements:

F1 User management - login with credentials (email, password), simplified registration with email only, users can see their profile information and log out.

F2 Content navigation - users can navigate the content structure, they will see which lessons are available and which are locked for them (because they do not have a valid license).

F3 Lesson display - all lesson content must be displayed, including pictures, sounds, diagrams, and equations.

F4 Questions and answers - all question types must be supported and after evaluation, correct answers and hints must be shown.

F5 XPs and diamonds - XPs and diamonds will be awarded and displayed. Users can see the level they achieved in each module.

F6 Friends leaderboards - users will see their progress compared to their friends.

F7 Friends management - users can find, invite and connect with friends

F8 Offline content - users can save lessons to use without an internet connection. This is a crucial part of the app, because makes it possible for students to study on the go, for example on the subway.

1.5.2 Non-Functional Requirements

Hrave API - the application will work with the current Hrave server through existing API

Android platform - the application will work on Android OS with minimum supported API level 15 (version "Ice Cream Sandwich") on both tablets and phones

iOS platform - the application will work on iOS 7 and above on all iPhones and iPads.

Offline mode - the application must handle changes in internet connectivity

Usability - the application should conform to the norms on the given platform and follow basic usability principles

Chapter 2

Design

2.1 Domain model

To have a clear grasp of the system, it is useful to describe all the entities in the system and their relations. The domain model for Hrave with all the relevant entities is shown in figure 2.1. Some properties are omitted for clarity. Next I will describe important entities in the system.

Domain Hrave is a generic educational software. A concrete deployment is called a domain. The user is tied with a single domain, however a real person can have accounts on more than one domain and have the same (or different) credentials on all of them. Hrave.cz is a domain. A Domain has a root entry, that is the root of the lesson tree (which typically contains modules).

Entry is any node in the lesson tree (see 1.1). The access type defines if it is free for users or if it must be unlocked by a license.

Module is the root of the lesson tree. At Hrave.cz these are subjects like Math and English.

Section is another layer in the lesson tree. There can be an arbitrary number of them. They can contain other sections or chapters.

Chapter is a section that contains lessons.

Lesson is an educational unit, designed to teach a specific topic. It consists of content parts.

ContentPart is a part of a lesson. It can be a non-interactive text (explanation) or a question part. Question parts contain a task and interactive question (depending on the type).

License Each entry (module, section, chapter, lesson) can be free or it can require a license. Users can have multiple licenses.

2.2 Use Case Analysis

Use cases are very useful by listing concrete steps that realize a certain goal in our system. Every functional requirement should be covered by at least one use case. The use cases might be altered when internet connection is not available - I will call this the offline mode.

UC1 Sign in is required for the user to use access the content. Sign in will be required only once, afterwards it will be automatic. In offline mode, the user does not need to (cannot) sign in, in order to view his saved lessons. If there are no saved lessons, no action is available.

1. The user taps the Login button.
2. Application shows the input boxes for email and password.
3. The user enters credentials and presses the Login button.
4. If the data is invalid, a dialog describing the error is displayed (e.g. empty email / wrong credentials / server cannot be reached). If the data is valid for more than one domain, the user is first asked to choose a domain. Otherwise the user is logged in - the main screen with modules is shown.

UC2 Register is possible with email only. After successfully registering, the user can login with new credentials. This is not possible in offline mode.

1. The user taps the register button.
2. Application shows the input box for email, a checkbox for agreeing with terms and conditions, and a link to see the terms and conditions.
3. The user inputs an email address and can optionally view the terms and check the checkbox.
4. The user clicks on the register button.
5. If the checkbox is unchecked, a warning is displayed, otherwise the user email is sent to the server and a response is displayed (e.g. user registered / email already in use / email invalid).

UC3 Sign out is possible if the user is signed in.

1. The user taps the logout button
2. The screen with sign-in and register buttons is shown

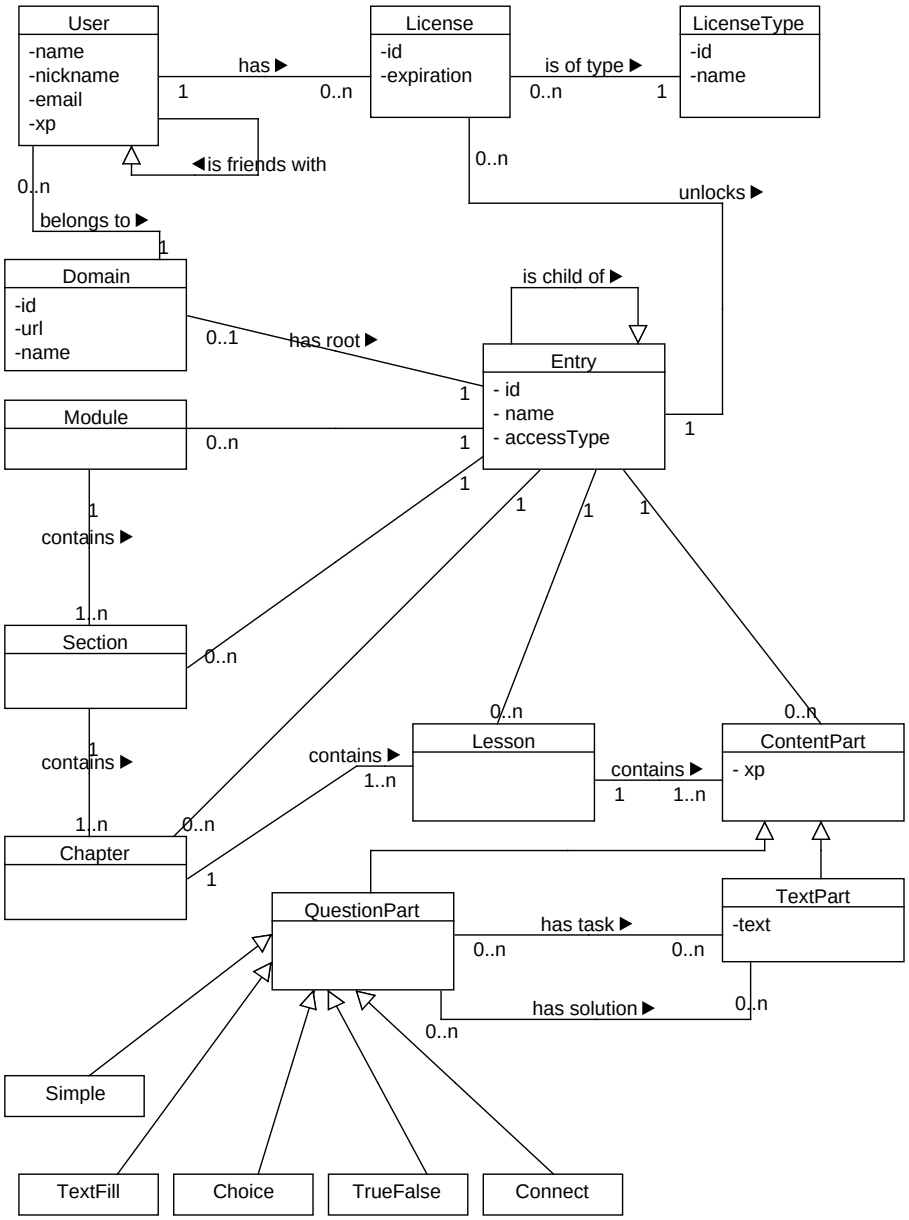


Figure 2.1: Hrave Domain Model

UC4 View profile information The user can view information about their account, including which licenses they purchased. This information can only be edited on the web. Not available in offline mode.

1. The user taps the user button
2. The screen with information is shown

UC5 Display modules Modules constitute the topmost navigation in the application. They are displayed in the main menu, which is only accessible to logged-in users. Works the same in offline mode.

1. The user navigates to the main screen.

UC6 Display section A section is any node in the lesson tree (see figure 1.1) that is not a module or a chapter. A section may contain other section or chapters (but never a mix of these two). In offline mode, items that are not cached or downloaded are disabled. Items that are locked for the user (requiring a license) are displayed with a lock.

First scenario

1. The user enters a module
2. Application shows a list of sections

Second scenario

1. The user selects a section or chapter
2. If it is locked, the application shows an information message
3. Otherwise the application shows a list of chapters

UC7 Display lesson list is analogous to previous case.

1. The user enters a chapter
2. Application shows a list of lessons

UC8 Play lesson Lessons comprise the core of the application and they are the place where users will spend most of their time. Lessons contain an arbitrary number of parts (either topic explanations or questions). In offline mode - **UC15**.

1. The user enters a lesson.
2. Application shows the first part of the lesson.
3. The user reads the text or answers a question and taps the continue button.
4. If the previous part was a question, the application shows the evaluation of the question. The application shows the next part.

5. The last two steps continue until there are no more parts to show.
6. Application shows the evaluation for the whole lesson including XPs.

UC9 See progress Users can see how far they progressed in a given module, chapter, or lesson. For chapters and lessons, this has the form of diamonds (silver and gold). For modules, they can see their XPs and level. In offline mode, progress can still be seen reflecting the last time the user was online. It might not be accurate, because the user might play on the web in the meantime, altering gaining XPs.

First scenario

1. The user enters a module
2. Application shows XPs and level.

Second scenario

1. In a section, the application shows the progress next to each chapter or lesson in the list.

UC10 Show friends leaderboard For each module, there is a leaderboard where users can compare their progress with friends. Adding this as a special screen would make it difficult to discover, but putting it inside the module would take up too much space. It was decided to make it appear after the user taps the friend icon in the header or taps their module progress panel. It appears below the progress panel, sliding down from it. If the user has not connected with any friends, a message is shown instead of a list, encouraging the user to find and connect with friends. Not available in offline mode.

1. The user enters a module
2. Application shows a list of chapters and above them it shows the user progress for the module.
3. The user taps the friend icon or the progress panel
4. Application shows the leaderboard for the selected module

UC11 Invite friend To compare scores with a friend, it is possible to invite them via email. If the email is associated with a user in the domain, a friend request is sent, otherwise an invitation email is sent. Not available in offline mode.

1. The user enters a module
2. Application shows a list of chapters and above them it shows the user progress for the module.

3. The user taps the friend icon or the progress panel
4. The user taps the invite friend button

UC12 Send friend request To compare scores with a friend, the two users must first connect. This is done via friend requests. If the email is known, the invite friend option is available. Otherwise the user can find their friend by their nickname. A minimum of three characters is required for a nickname search. If a user is found that has sent a friend request, user can accept it in the search results. User already in friend list are not included in search results. Not available in offline mode.

1. The user enters a module.
2. Application shows a list of chapters and above them it shows the user's progress for the module.
3. The user taps the friend icon or the progress panel.
4. The user taps search button.
5. Application shows a search dialog with an input field.
6. The user inputs at least three characters and taps 'search'.
7. Application shows a list of users.
8. The user taps the send friend request button next to a selected user.

UC13 Accept/Refuse friend request Friend requests can be accepted or refused. Use cases are very similar.

1. The user enters a module.
2. If there are any friend requests, the friend icon in the header is highlighted, if there are two or more, the number is shown.
3. The user taps the friend icon or the progress panel
4. The user taps the friend request button (which also shows the number of requests)
5. Application shows a list of friend requests.
6. The user taps the accept or refuse button next to a selected request

UC14 Save lesson offline To keep things simple, only whole chapters containing lessons can be downloaded for offline use. This can be done by pressing a button in the top panel inside the chapter. A download is only available if there is active internet connection and if the chapter was not yet downloaded. Not available in offline mode.

1. The user navigates to a chapter containing lessons.

2. Application shows a list of lessons and a download button in the panel.
3. The user taps the download button.
4. Application shows a progress bar.
5. Application finishes the download and UI changes to reflect that lessons have been downloaded.

UC15 Play lesson offline This is similar to UC8, however, only downloaded lessons are available.

UC16 Resume previous When the application is closed during a lesson or deeper in a lesson tree navigation, the position gets saved. It can be restored after the application is restarted. A saved position older than 4 days is considered irrelevant. Lessons are always restarted from the beginning.

1. Application shows a list of modules with a continue button next to previously closed module.
2. The user selects the continue button.
3. Application shows the previously closed lesson or section.

	Requirements							
Use Cases	F1	F2	F3	F4	F5	F6	F7	F8
UC1	■							
UC2	■							
UC3	■							
UC4	■							
UC5		■						
UC6		■						
UC7		■						
UC8			■	■				
UC9					■			
UC10						■		
UC11							■	
UC12							■	
UC13							■	
UC14								■
UC15								■
UC16		■						

Table 2.1: Use case coverage of functional requirements

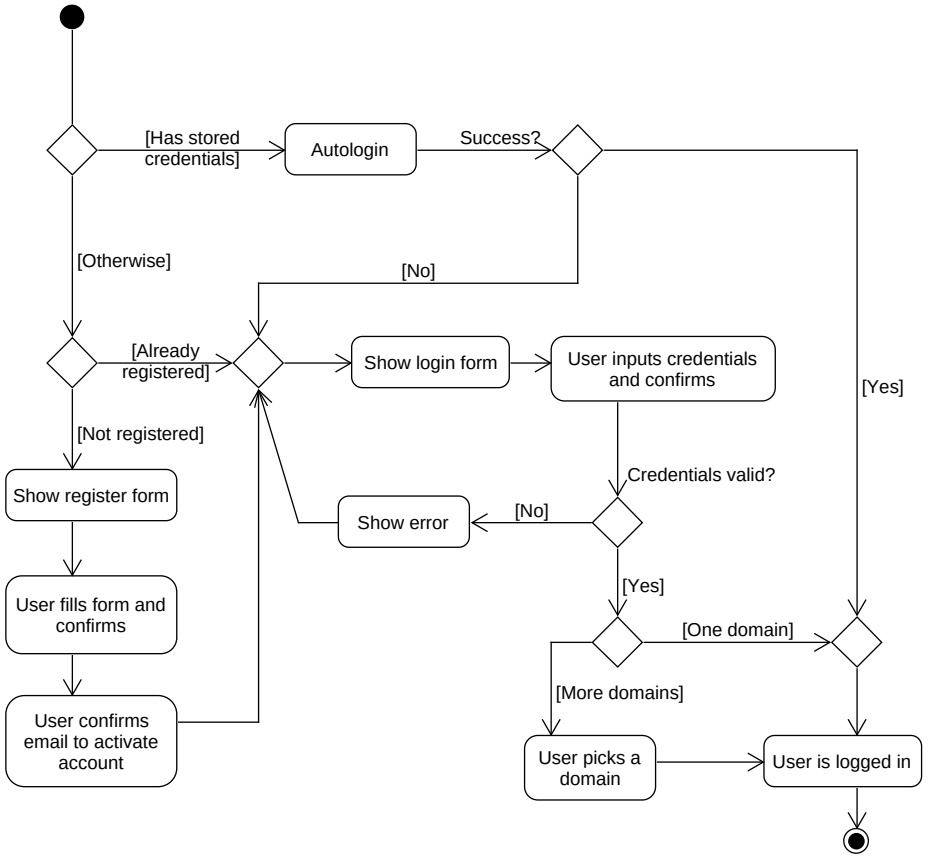


Figure 2.2: Login Activity Diagram

2.3 Activity Diagrams

Use cases and user scenarios are immensely helpful for understanding the system from the user point of view. To better understand certain scenarios from the system point of view, activity diagrams can be used. In figure 2.2, you can see a login activity diagram and in figure 2.3 the activity in a lesson.

2.4 Hrave API

To communicate with the server, this mobile application will use the same API the javascript frontend is using. The API is not stateless, the user is identified by a session ID which is tied with a state on the server, most importantly to

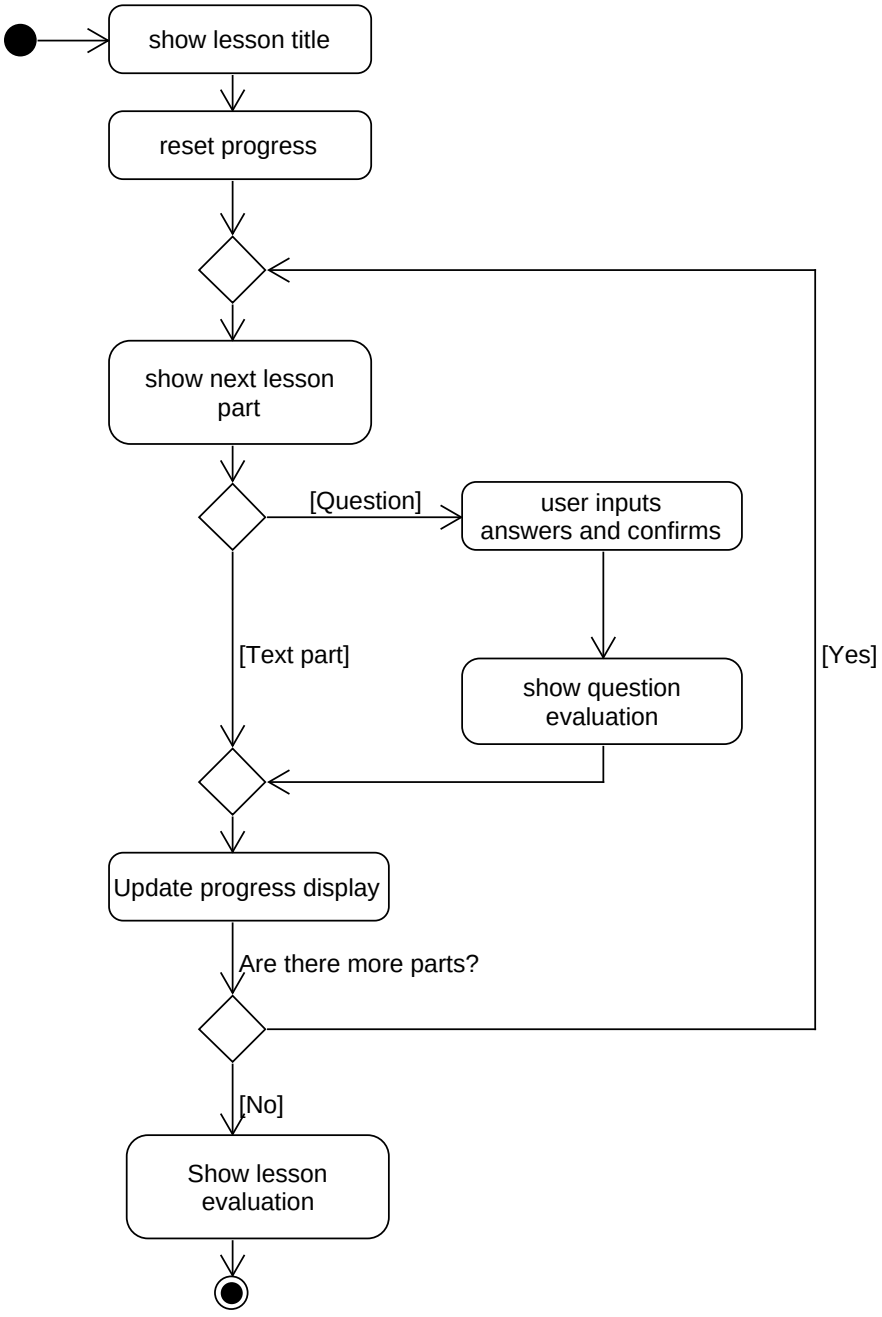


Figure 2.3: Lesson Activity Diagram

the user and which lesson (if any) is opened. On the web, the sessionID is saved in a cookie object. On mobile, this is not an option, so it is appended as an argument to any URL request. All requests are sent over HTTP using the POST method. Data are encoded with JSON.

These are some of the relevant parts:

/loginSession.do This request takes the parameters login, passwd, domainId and returns a new session ID that should identify all future requests.

/getUserDomains.srv This request takes the parameters login and passwd and returns all domains that have a user that has these credentials.

/logoutSession.do This request simply ends a session. The session ID can be discarded.

/registerMail.srv This request takes an email and domainId and creates a new user on the server. It sends a confirmation email to the user.

/openDisplayObject.srv This request takes an id parameter, which is a lesson to be opened. The user can only have one lesson opened at a time. This is mainly to prevent students from sharing a license.

/evaluateAnswer2.srv Takes the parameters qId and answerJson. Returns the evaluation of the answers as well as possible additional text parts as explanation. Also saves the result, for final lesson evaluation. If it is called with ID that belongs to a question in a lesson that is not opened in this session, it returns a error.

/closeDisplayObject.srv Closes the lesson, so no more answers can be evaluated in it.

/getECResults2.srv Returns a lesson evaluation, based on all question evaluations. Allots XPs to the user.

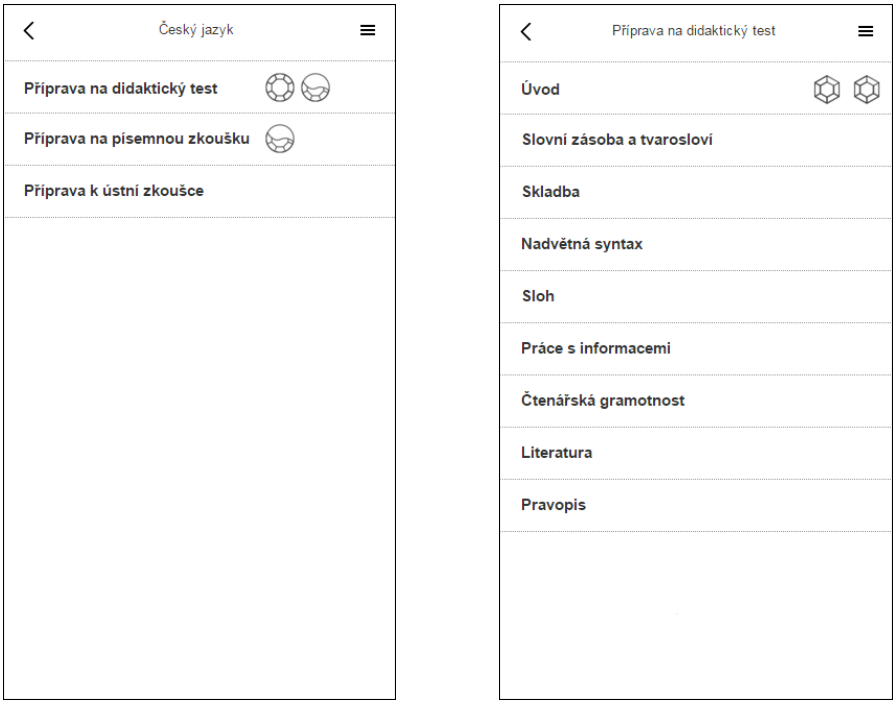
/getPageData.srv This request takes the id of an entry and returns all information including XPs and children objects. This is used in every step, when the user is navigating the lesson tree.

/friendsActionEndpoint.do This request is used to manage and interact with friends. It takes an action parameter and possible other data depending on the action. The action can be one of the following: getFriendRequests, emailFriendRequest, friendFindByNick, friendRequestSend, friendRequestAccept, friendRequestRefuse. These action names should be self-explanatory.

2.5 Wireframes

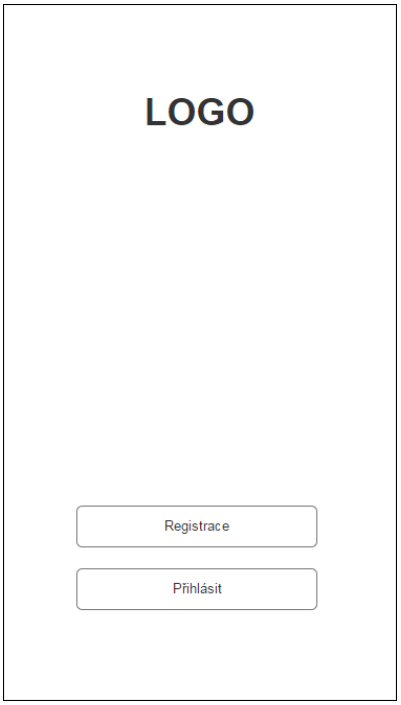
At this point in the development, I began to work with the graphic designer at Educasoft, Petr Miloš, on the concrete look of the app. This next chapter therefore comprises the result of the collaboration, where we combined my knowledge of mobile systems and Petr's experience with graphic design, UI, and user experience (UX). We used the use cases as a starting point. Some features were redesigned in the final application, for example, there is no 'menu' button in the header.

The flow of the screens is shown in diagram 2.6.



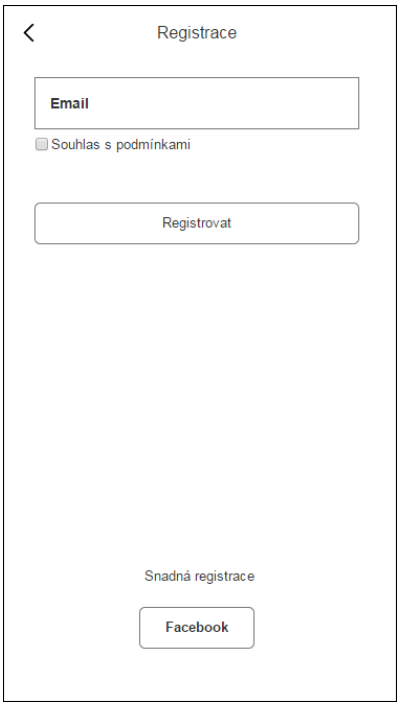
(a) Section Screen (b) Section Screen

Figure 2.4: Wireframes I.



(a) Start Screen

(b) Sign-in Screen



(c) Register Screen

(d) Modules Screen

Figure 2.5: Wireframes II.

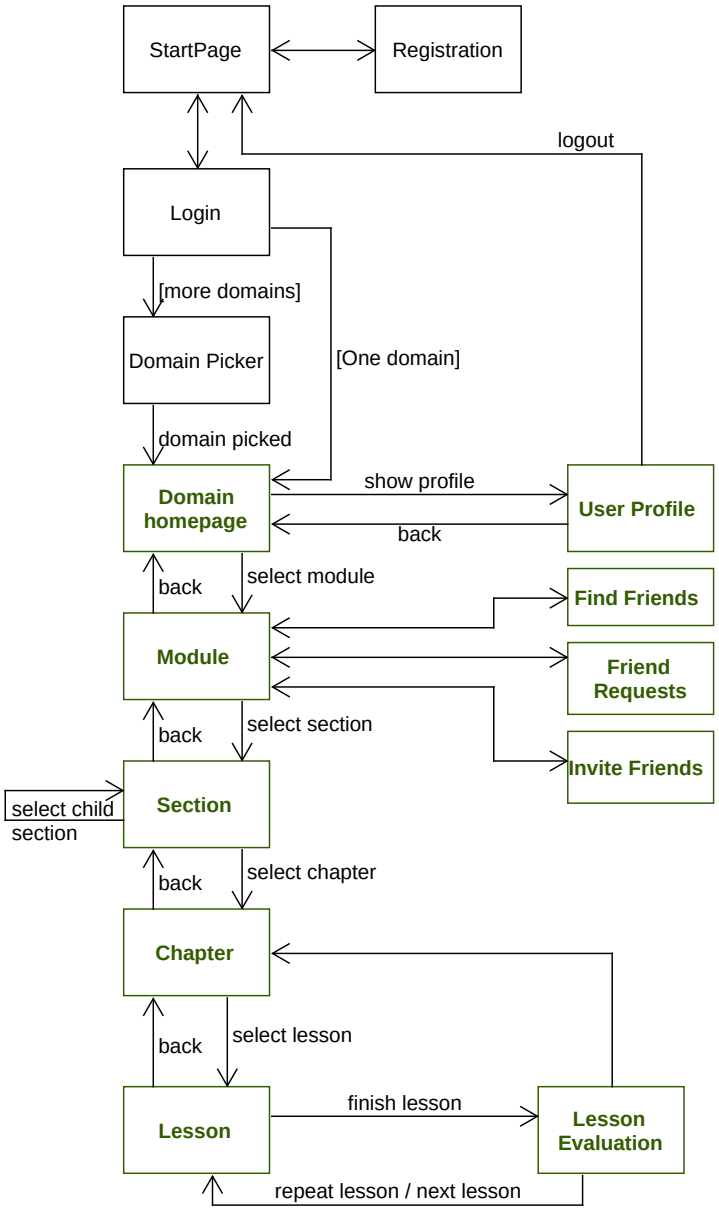


Figure 2.6: Screen Flow

2.6 UI

2.6.1 Navigation

Navigation in mobile applications is often solved by a left navigation drawer menu that is opened either by a swipe gesture or by tapping the so-called 'hamburger icon' in the header. This solution allows for substantial number of options, but it also has significant disadvantages. It has low-discoverability, requires two taps for any navigation, clashes with the back button in the header, and it is not glanceable (e.g. cannot notify the user that there is a new message in their inbox by placing a number next to the inbox button, because they will not see the button unless they open the drawer)[28].

The solution for Hrave was to place the action buttons directly in the header. Since accessing the user profile is not a very common action, its button was placed only at the top navigation screen with modules. Any other screen will have a back button on the left, so there is only space for buttons on the right. Inside a module, we can see the 'friends' button, which opens the leaderboards for this module and buttons for friends management. When the user has a friend request, it shows the number on top of the icon to draw attention to this fact. In a chapter, there is a download button, and in a lesson, there is a button with a diamond that opens and closes the lesson progress display. In the offline mode there is a button on the right, indicating the offline state. When tapped, it opens a dialog explaining the offline mode with the option to try to reconnect. Friends and download buttons are not visible in offline mode.

With this design, there are no menus. All the items are clearly visible, when appropriate and it fits with the design guidelines on both platforms. See figure 2.7 for all the actions in headers.

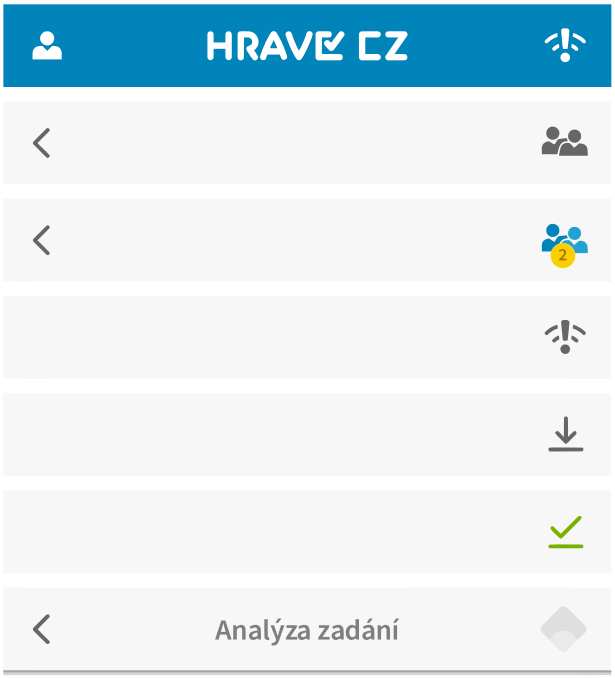


Figure 2.7: Header Actions

Chapter 3

Implementation

I had access to the source code for the javascript frontend which I used as documentation for the server API and other features. Some parts could be ported to AS3, like lesson content parsing, most of the other things however had to be built from scratch to ensure modularity, readability, and extensibility of the new code.

3.1 Tools

To manage the implementation process, I used several handy tools. For bug tracking and task management I used Asana[29] which was already in use by the Educasoft company. It includes all the usual features, while being very easy to use.

To track the time I spent on implementation, I used Toggl, which is an online time tracker with reporting features.[30] I found it very easy to use.

3.1.1 Versioning

For source-code management and versioning, I used Apache Subversion (SVN). I added a new project folder to the existing repository running on Educasoft servers. For my purposes it was convenient, thanks partly to the TortioseSVN client application. If it was not for the fact that it was the established version control system (VCS) at the company, I would consider using a decentralized VCS like Git or Mercurial. In these systems, each checkout is a full repository with a complete commit history and they offer easier branch control[31]. For a single developer with a fairly short development time, SVN was satisfactory and I did not encounter any issues. I usually committed all changes after each implemented feature or bugfix.

3.1.2 IDE

To develop software with Adobe AIR there is a choice of integrated development environments (IDEs). We could, of course, use any text editing software, but an IDE will make development faster, enable quick refactoring and easy debugging.

Popular choices for an IDE include the Adobe Flash Builder, IntelliJ IDEA and the free and open-source Flash Develop. I decided to use Flash Builder, which is built on the Eclipse platform and features an interactive debugger and the Android USB debugging feature.

3.2 Feathers SDK

Feathers UI is a collection of components built upon the Starling framework, that supports many commonly used controls and layouts and is fully skinnable. Feathers SDK is a new free and open-source tool from Bowler Hat. Firstly, it simplifies the process of managing different versions of Adobe AIR SDK, Starling, and Feathers, with the SDK Manager program. Most importantly, it allows developers to statically declare components in MXML files. These were originally used with the Flex framework, but they were remade to work with Feathers and therefore deliver a great performance on mobile[26].

The following code example shows how it is possible to reference declared components and mix declarations with code. The code example creates an application with a slider with values from zero to a hundred, a label that always shows the current value, and a button that prints the current value to the console.

```
<f:Application xmlns:f="library://.feathersui.com/mxml"
       xmlns:fx="http://ns.adobe.com/mxml/2009"
       theme="feather.themes.MetalWorksMobileTheme">
       <f:LayoutGroup>
              <f:layout>
                     <f:HorizontalLayout padding="10"/>
              </f:layout>

              <f:Slider id="slider" minimum="0" maximum="100" value="10"/>
              <f:Label text="{slider.value}"/>
              <f:Button triggered="button_triggeredHandler(event)"/>
       </f:LayoutGroup>

       <fx:Script>
              <![CDATA[

              private function button_triggeredHandler(event:Event):void
              {
                  trace( "slider value changed! " + this.slider.value );
              }

              ]]>
       </fx:Script>
</f:Application>
```

This is similar to Android development, where layouts and visual elements can be declared in XML files[32], however, it differs in several aspects: Let me demonstrate on a simple example, with a button and a function that is called when it is pressed.

In Android, we can declare the button and set a unique ID, because we need to reference it later.

```
<Button android:id="@+id/my_button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/my_button_text"/>
```

Then in our code we need to get the reference to that button by calling a function:

```
Button button = (Button) findViewById(R.id.my_button);
```

Then we can attach a listener to it.

```
button.setOnClickListener(new View.OnClickListener() {
       public void onClick(View v) {
           // Perform action on click
       }
});
```

It is also possible to assign a method directly to the button in XML, but it has to be in the activity class and be public[33]. The popular library Butter Knife makes it possible to use annotations with IDs to automatically bind views from a layout to fields in a class,[34] this however makes it necessary to

declare the variables as fields, even though they might only be needed locally.

Now in Adobe AIR with Feathers SDK we declare our button and we can attach a listener function directly. AS3 has function references, so this could be any private or public method.

```
<f:Button id="my_button"
       label="my_button_text"
       triggered="onClick(event)" />
<fx:Script>
       <![CDATA[
       private function onClick(event:Event):void
       {
           // Perform action on click
       }]]>
</fx:Script>
```

If we wanted to manipulate the button in the program, we could reference it just by typing its ID, because the compiler has information about declared components in the MXML file. It is even accessible from outside of the MXML file just like a public variable. This saves us from writing a lot of boilerplate code.

One thing to note here is that Android has great localization support and all the strings are declared in a strings.xml file and referenced by the *@string* notation. In section 3.9, I describe a custom localization mechanism.

In conclusion, in both Android and Feathers, declaring UI elements and layouts makes code much more readable. Because the components in Feathers MXML are accessible directly from the code, they are easier to work with.

3.2.1 Application Descriptor File

Each Adobe AIR application has a descriptor XML file, which includes essential information like display name, version, and icons. Depending on target platforms, it can also include Android Manifest additions like permissions and iOS info additions.

3.3 RobotLegs

To organize the code, I used the RobotLegs framework[35]. Firstly, it provides a dependency injection mechanism. In common object oriented programming, one object often needs to access the functionality of another object. Normally, the object would itself be responsible for getting the reference needed. With dependency injection however, it can just declare the dependency and it will be automatically injected by the framework, which allows for loosely coupled solutions that are easier to test[36]. RobotLegs uses annotations to declare dependencies.

For example:

```
[Inject]
public var myDependency:MyClass; //unnamed injection
```

The class that handles injections is an Injector and it is available in a Context class, which is a central part of every RobotLegs application. To map a singleton to the aforementioned dependency we can call:

```
injector.mapSingleton(MyClass);
```

There are, of course, more options. We can map a new instance of MyClass every time it is needed or map a specific instance (not a singleton) or use injection in the constructor or with a setter function. We can declare dependencies with interfaces and then inject concrete implementations, which is ideal for testing. Dependencies can also be named to provide additional information[37].

3.3.1 MVC

RobotLegs is built around the MVCS architecture pattern. This is like the classic Model-View-Controller (MVC) with the addition of a fourth actor called Service. The goal of this pattern is to promote loose coupling, organization of code, and reusability[38]. Importantly, the architecture is very loosely enforced, so it is possible to use only parts we need and organize other parts differently.

3.3.1.1 View & Mediators

With Feathers SDK the UI components are defined in MXML files and ideally they should not contain any application logic. For that they are coupled with a Mediator. The coupling is defined in the Context class.

```
mediatorMap.mapView( viewClassOrName, mediatorClass );
```

A Mediator is created automatically when the view is placed on stage (becomes visible). The View listens to user actions and dispatches events. The Mediator handles these events and responds according to the application logic[38]. This means that views have no dependencies and are easily reusable.

One example of this - when we agreed with the graphic designer that the login screen should not be a whole screen, but merely a panel that slides in from below, I only needed to slightly change the view, but the mediator required no change at all, and all the application logic stayed intact.

3.3.1.2 Commands

Commands represent the controller tier in RobotLegs. They are designed to perform a single task in the application. They are invoked by events and can

perform actions on other actors in the system. They are usually created in response to an event, they perform a task, and are immediately discarded.

3.3.1.3 Model

Models should encapsulate and provide access to data in the application. They can send event notifications about data changes.

3.3.1.4 Service

The last of the MVCS actors is Service, which should provide an API for communication with some outside entities like web services and file systems. Typically, we should ask a Service for some resource and then listen for events that signal the success or failure of the operation. Purely out of convenience, I used callbacks for this purpose, so when we call a function on a Service, we also supply a function to be called for success and for failure (error).

3.4 Package Description

The package structure loosely copies the MVCS architecture.

Command contains Command classes, for example UserLoggedOutCommand, DeactivateCommand.

Event contains custom Events.

Mediator contains Mediator classes for each view. Typically, there is one mediator per screen view, plus there are some subcomponents that have their own Mediator.

Model contains Model classes. The application has one central model class - HraveModel, that takes care of loading and caching the lesson tree and content. Also in this package there are the value object (VO) classes, simple, strongly typed classes used to store information retrieved from the server, so that it does not have to be passed around in dynamic objects parsed from JSON.

Service contains Service classes.

Util contains usually portable utility functions.

View contains all the MXML files, defining the Views.

3.5 Offline Mode

The application must work both with and without an active internet connection. This is a global state that influences behavior across all components. The app monitors the connection and reacts appropriately. There is a NetworkInterface API in Adobe AIR that makes it possible to find out if the user has active wifi or mobile internet. However, practically, it does not guarantee an internet connection. Therefore I only listen for the 'network_change' event, in reaction to which I perform a simple internet request and wait for the HTTP status 200 (OK) or an error. If we lose internet connection, an event is dispatched through the application. There is a special ConnectionLostCommand (see section 3.3.1.2) that handles this situation.

If the user has any saved lessons and loses connection, the application simply lets him browse all his downloaded content. If there are no saved lessons, the application will not allow the user to continue until there is a connection (there is a 'retry' button).

3.6 Graphics

Mobile applications must deal with a vast range of devices with wildly different screen sizes and resolutions. A strategy is needed to assure a usable and clear graphical interface with all of these devices. A naive solution is to design everything for a specific resolution and then either stretch the graphics canvas to fit the whole screen or to leave black stripes around it (this is called letterboxing). This gives us either blurred content or if we choose a very high target resolution, the controls could end up being too small to be usable. I will now describe the approach I used.

I took the concept of the density independent pixel (dp) unit from Android.[39] This is a virtual pixel unit that will stay approximately the same physical size on the screen independently of its resolution and size. It is equal to one physical pixel on a 160 dpi (dots per inch) screen. If we define the dimensions of our UI controls with this unit, we do not have to worry about it being too small to touch or too large on devices with large screens like tablets. I can calculate how many actual pixels correspond to 1 dp on the current device as $device\_dpi/160$ and I call this the dp scale.

Starling is built upon the Stage3D API, therefore it uses the GPU to display graphics. All graphics must be first converted to textures to be displayed. The graphic designer prepared all references and assets in vectors and sized them as if for a device with 160 dpi. I resized the assets by 200% and then packed them into a texture atlas.

To use a texture atlas is a common optimization technique used by any engine working with the GPU. Because it is costly to swap textures on the GPU, we pack all the small textures into a large one and add a XML descrip-

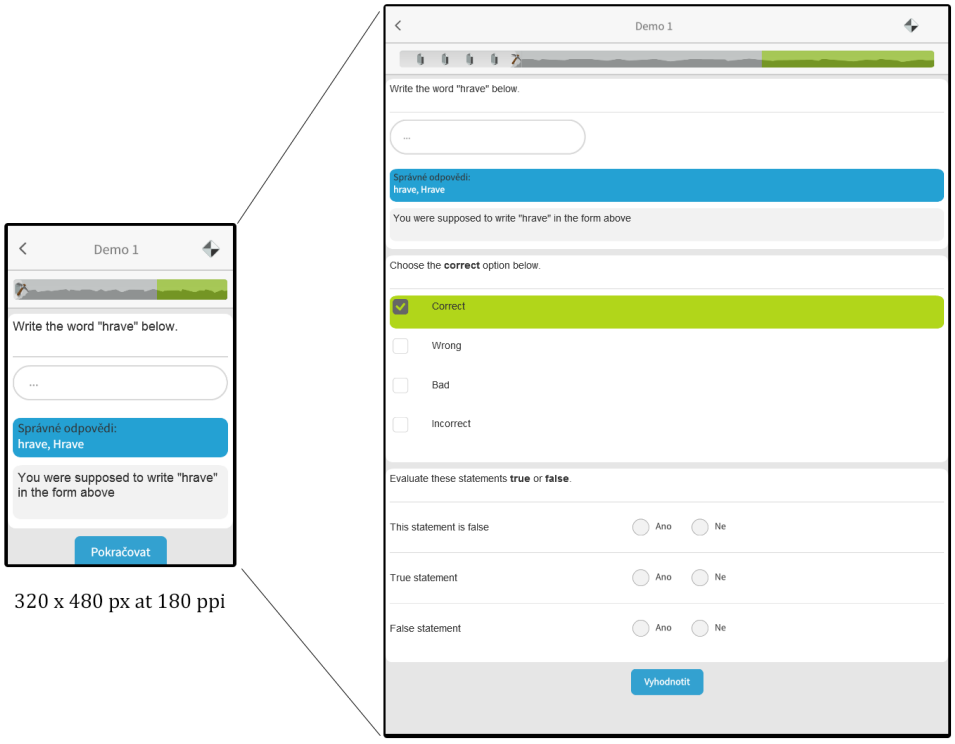


Figure 3.1: Multiple Screen Support

tion of where the smaller textures are to be found. This means that during runtime there is no need to change textures for each image, we can simply supply different texture coordinates to the GPU[40]. Starling has a built-in support for texture atlases.

It also offers the possibility to pass a scale parameter with a texture, so when we pass the texture atlas, I supply a scale of 2 to account for my upscaling. This way I can measure everything with dp units and when I get a texture I can simply scale it up by the dp scale to get the intended size. With this technique, relative positioning and layouts, everything looks as close as possible to the graphic design, and all controls have reasonable sizes on all devices. Testing showed that the textures are sufficiently crisp even on high density screens. In figure 3.1 the same lesson is shown on two very different devices. We can see that on tablets the extra space is used to display more content.

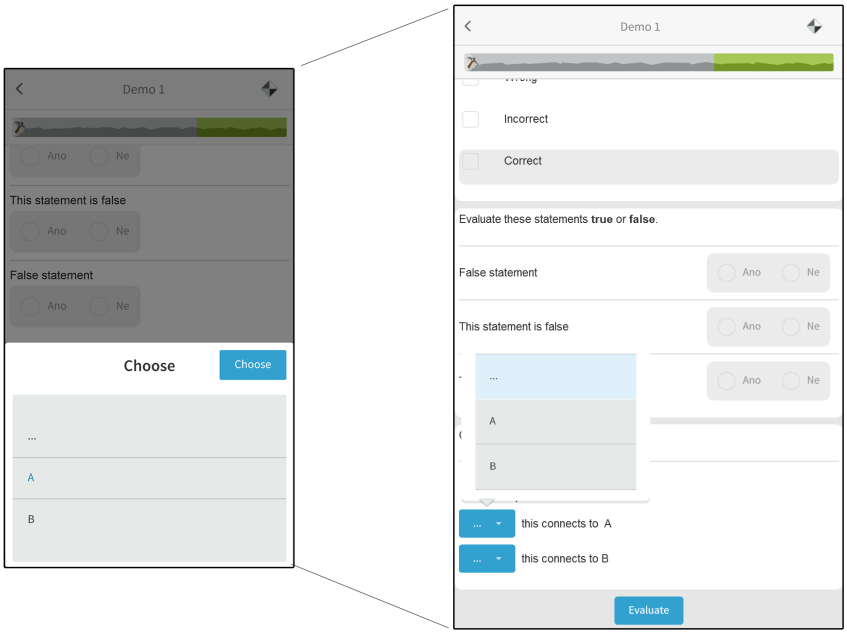


Figure 3.2: TextFill question type of phones and tablets

3.6.1 Controls

Feather's themes make it possible to completely separate logic from appearance. Similarly to CSS, it makes it possible to skin any component of a certain type or all components with a specific 'styleName'. The component is then passed as an argument to a style function we provide, so that the skin, dimensions, and other properties can be set.

To quickly start prototyping it is easy to download and integrate one of the example themes and gradually override some of its functions.

With the Connect question type (see figure 1.2), the user has to pick from a set of options. This is usually done with a Picker control on mobile. It is not possible or necessary to show all options all of the time, therefore just a button is shown with an arrow to indicate there is a list of options and when it is pressed, a picker is shown for all options across the screen. With Feather's themes it is possible for one component to behave differently on tablets and phones. I utilized this feature and on tablets the options are shown on top of the pressed button (see figure 3.2). The same happens with TextFill questions.

3.7 Content Parsing

The content for the lessons is created with an WYSIWYG HTML editor, therefore an important part of the implementation was to try to display everything as closely as possible to what is rendered on the web. The AS3 has only a limited support for HTML and CSS. It supports some basic formatting, but on mobile it does not support links, images, or tables. To work around this constraint, I firstly had to parse the HTML to make it easier to work with. I used the open-source AS3 HTML Parser Library by Ryan Groe[41]. It was clear that I would not be able to render the unsupported elements inline with the rest of the content, however, since the horizontal space is very limited on mobile anyway, I can put these special elements on a new line of their own.

The process was to go through the HTML tree and take out the special elements and render them once all the previous tags were closed. The rest of the formatted HTML must also be fixed to account for the messiness inherent in a WYSIWYG system like empty tags and multiple newline characters.

3.7.1 Images

Images in HTML are not supported in Adobe AIR on mobile platforms. Implementing support for ordinary images in png or jpeg format was very straightforward. However, for equations and other vector images the editor supports the scalable vector graphics (SVG) format. Images in this format are described with XML as a collection of objects (lines, fills, text objects, etc. . .)[42]. It is possible to use SVG in Adobe AIR, but it has to be embedded at runtime. For dynamic SVG, I found an open-source library called Svgweb which was originally developed to display SVG in web browsers using the Flash plugin[43]. It creates the vector image by utilizing the Flash display graphics API that can be used to create vector shapes[44].

To use these images in Stage3D they have to be rasterized, which is possible, because every object in Flash display list can be drawn into a Bitmap[45]. The advantage of this is that we can create the vector image in the target scale, so that it will be perfectly crisp even on high density screens regardless of the original dimensions on the web.

After testing the images in Hrave lessons, I found the library to be working mostly correctly, however for a few images the rendering just failed and rendered nothing at all.

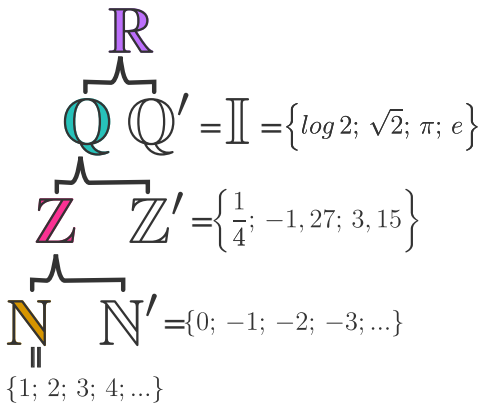


Figure 3.3: Hrave SVG Example

I fixed a minor issue in the source for one specific image, but so far the only solution for other images was to recreate them with simple elements in SVG or to convert them to some raster format.

Another issue is that some lessons contain inline equations with text right before and after. Therefore with my solution there is a gap in the text and subsequently there is the image. One solution is to adjust all lessons with this restriction in mind. A more complicated solution would be to try to emulate the HTML rendering process.

3.7.2 Tables

To recreate tables I utilized the Feathers layouts. The table itself is a container with a vertical layout and for each row I create a new group with a horizontal layout. For each cell I create yet another group and render the inner HTML inside (this is recursive, so there can be text, images, even other tables). The only problem is setting the dimensions of the cells and the rows. Fortunately, I found that the dimensions are mostly set in the HTML 'style' property, therefore I could use it directly. If the cells have no width property, the width of the table is divided equally among them. If the table is wider than the current screen, it is compressed to fit.

3.7.3 Links

The HTML text can contain links. These can be links to anywhere on the web, but often they reference parts of other lessons like definitions or theory parts. On the web they open the content in another window in a special 'view-only' mode.

On mobile I added them to a list at the end of the lesson. If the link targets a lesson part, it is opened in a full-screen window inside the app. The lesson part may contain links as well, so these windows are stacked on top of each other. External links are marked with a special icon. If the user taps on them a dialog is shown, warning the user that they are about to leave the application. They open the default internet browser.

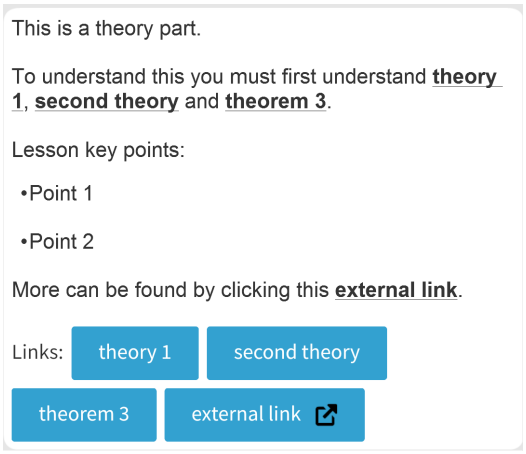


Figure 3.4: Example of links in a lesson (mobile app)

3.7.4 Videos

Videos from YouTube can be embedded in a lesson. To support these lessons I simply added another type of link at the end of the lesson, which opens the video in a YouTube app (if it is installed on the device). On Android, the user can navigate back using the back button. On iOS they have to go back either by pressing the home button and opening the app again or through the list of opened apps (double pressing the home button). When users taps the video link, a dialog is displayed, warning them that they are about to leave the app and have to return to continue the lesson after they finish with the video. The disadvantage of this approach is that, if the device is low on memory, it may dispose of the Hrave application that is running in the background when the video is played. When the user returns, the lesson has to be started from the beginning.

3.8 Sound

Some lessons contain sounds, for example some English lessons contain listening exercises. The main problem here was that all the sounds on the server are saved in a Ogg Vorbis format, which is not supported by the Adobe AIR runtime. Android however supports Ogg Vorbis streaming through the MediaPlayer[46], so I decided to use an ANE to implement this functionality. I found an open-source ANE for accessing the MediaPlayer from FreshPlanet[47]. However it lacked functions for stopping and resuming the sound and most importantly, it would not compile with my application. I decided to create my own ANE based on this one.

There are several steps needed in order to create an ANE for Android:[48]

1. To create the Android functionality, the Android SDK has to be set up.
2. We create a new Android project and add two JAR files from the Adobe AIR SDK. These will allow us to use special wrapping objects to transform data from Java value types to ActionScript value types.
3. We create a class that will act as an interface between the ActionScript and Java code. It will implement the FREExtension interface.
4. For each method we would like to use, we create a special class that implements FREFunction interface. In the extension class we will create a method that takes a function name as a parameter, and returns an instance of the correct FREFunction class.
5. Export the Android project as a JAR file.
6. Now for the ActionScript part of the ANE. We create a class that will create a context that will serve as a bridge between the extension and

the ActionScript program. Then we create a method for each function of the ANE and use the *extContext.call* method to delegate the work to the Java part of the ANE.

7. We export this part as a AS3 library.

8. Now we have a Android JAR file and the AS3 library. To combine them, we first need an extension descriptor in XML. There, we set an ID for the extension and specify the fully qualified class name of the FREExtension class from the JAR file.

9. With the command line tool adt from Adobe AIR SDK we then package these files into a single .ane file, which we can include in our application.

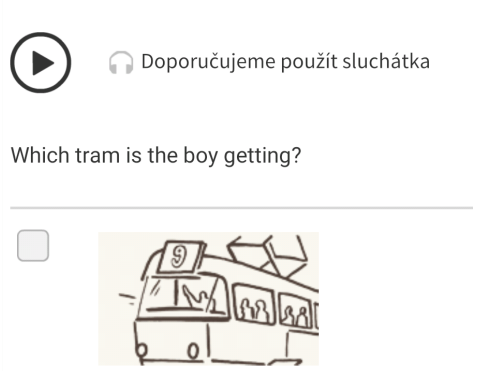


Figure 3.5: Sound Player on Android

This approach worked and I created a simple sound player component which worked on Android.

On iOS, however, there is no support for Ogg audio playback. The source-code for Ogg decompression is freely available, so one option for iOS would be to compile this code for iOS to play the decompressed audio with iOS Audio Queues Services[49] and bundle this into an ANE. iOS ANE can only be compiled on a computer running OS X, which was not available to me.

Another option is simply to keep all files on the server in the MP3 format. So far this has not been solved, so audio works on Android devices only.

3.9 Localization

Although currently *Hrave* is only used in the Czech Republic, it is prepared for any localization. To support this in my application, I, once again, took inspiration from Android[50]. All strings are defined in a string.xml file, that resides in a folder named as the locale shortcut (e.g. cz or en). This file is

loaded and parsed on startup by a singleton class Resources. When I need to use any text in the application I call the *Resources.string* method and put the string identifier as an argument. Optionally the string can contain a placeholder for a value, the following example shows the format:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
      <string name="friend_invite_sent">Invite sent to email %1$s.</string>
<resources>
```

Localization is used for texts in the application like button labels and alert messages. The content of the lessons is not affected.

3.10 Persistance

There are multiple instances of caching and persisting data in the application. Firstly, whenever it displays a module, section or a chapter the parsed data needed to display that screen are cached. Therefore when the user navigates through the lesson tree, the application only pauses to load once for every screen. When the user returns to it, it gets displayed without any loading. This is cached in memory, therefore it is not persisted between user sessions. The model keeps a tree of loaded entries, because when the user gains XPs in a lesson all its ancestors in the tree must be invalidated to reflect the change.

Secondly, when we receive data from the *getPageData.srv* request, they get automatically saved in a text form in a local database. Adobe AIR supports SQLite, the light-weight relational database engine[51]. The database itself is stored in a single file and data is accessed through SQL queries. The one in the Hrave application only has one table with three columns (see figure 3.6).

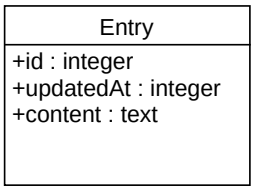


Figure 3.6: Database Schema

When the user downloads a chapter for offline use, first all the lesson data are requested from the server and saved in the local database, then all the lesson content is parsed and all the images are downloaded to the disk. A timestamp is also saved. In the future, there should be a mechanism that would update downloaded lessons, if the content changes. Also it should be possible for the user to delete downloaded lessons and see how much storage they occupy.

Lastly, all the other information that needs to be persisted between user sessions is saved with the SharedObject API. This is a simple mechanism in Adobe AIR to save simple objects under a name, similar to SharedPreferences in Android. Properties saved this way include the user login and the last opened location.

3.11 Performance Profiling

Mobile devices are still significantly less powerful than desktop computers and especially since I am not using native components, it is important to optimize performance to ensure a positive user experience. Adobe developed an advanced profiler for AS3 with CPU usage, function sampler, memory allocation tracking and even a detailed Stage3D usage analysis, including a command-by-command replay[52]. It can also be used on Android.

In development, I noticed a performance issue in going to previous screen using the back button. There would be a significant pause of up to 800ms between the release of the back button and the switching of the screens. I profiled it and found out that when the previous screen is recreated, it already has all the data, so it renders all the buttons at once. Each button has a label with the name of the section or chapter and the creation of these labels is what was causing the delay, the profiler revealed.

With Feathers, when we want to display text and we are not using a bitmap font, the text gets rendered by the Flash text engine and then it is drawn into a bitmap and uploaded to the GPU.[53] Doing this multiple times per frame is ill-advised. To fix this issue I added a small pause before each button is added to the list. Therefore as soon as we hit the back button, the screen transition begins. After that, one by one, the buttons are drawn on stage. This significantly improved the responsiveness of the application.

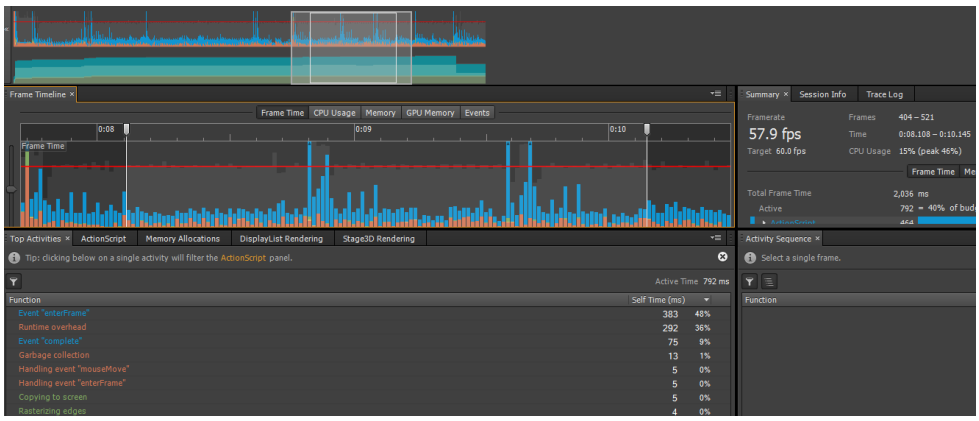


Figure 3.7: Adobe Scout

3.12 Login Security

When the application is first opened, it requires a login. After a successful login, it should not keep asking for it every time it is opened. The credentials must be persisted, however it would not be secure to store the user's password in the application storage, where it could be discovered and potentially exploited.

The solution was to send the password to the server only during the first login. Then the server returns a salted hash, which is then stored instead of the password. On subsequent logins only the hash is sent. This is still not ideal, but at least the original password will not be visible to an attacker.

Chapter 4

Testing

One of the advantages of using Adobe AIR was that I could run the code on a desktop computer without any emulator and quickly test new code. This way the majority of bugs were caught. Only after a feature was tested on desktop did I test on a real device. I had these devices available to me for development:

1. Samsung Galaxy S3
2. LG Nexus 5X
3. iPhone 4S

Testing on Android was easy, since Flash Builder supports Android USB debugging which means that it is possible to test and debug the app (with breakpoints and variable watches) on a real device via an USB cable[54]. On iOS, the application had to be first registered online with a Apple Developer account and then exported and signed with appropriate certificates and update through the iTunes software. For this reason, the iPhone was usually the last device to test on.

4.1 TestFairy

I periodically released new versions for both Android and iOS, so that testers in the company could try them and report any issues. For this, I used TestFairy, a free service for Android and iOS beta testing. Firstly, it handles the distribution of new versions to testers. It can email testers about new versions or automatically install a new version when the user opens the app.

Secondly, it gathers not only statistics and logs about testing sessions, but it also records the sessions on video and adds CPU and GPU performance graphs. From each testing session the developer can see information about

the tester and the device, the video of the screen, screenshots, logs and performance graphs. When something goes wrong, it is very easy to find out what caused the issue.

Thirdly, the service allows testers to submit bug reports straight from the app by shaking the device (this feature can be turned off). This can be connected with popular bug tracking software including Asana[55].

It is also possible to upload new versions automatically, which might be useful for continuous integration.

4.2 Usability Testing

After all main features were implemented, the application was tested on real users. The usability testing requires real world scenarios that are given to the user while the tester observes and takes notes. These scenarios should be realistic and not contain any hints as to how to accomplish given tasks.[56] The users were all Android users, so the application was installed on their own devices using TestFairy.

The registration process was not tested, as it was clear it is not yet well suited for mobile, because it currently requires the user to leave the application to confirm their email address and then come back. For the future, this will be replaced by Facebook registration and with email that does not have to be confirmed straight away. Users were therefore presented with login information as if they were already registered on the website, but had just installed the mobile application.

A short survey with these questions preceded the testing:

1. What is your age?
2. How would you describe your proficiency with Android? (novice, intermediate, advanced)
3. What Android phone do you currently use?

Then the application was installed on their device and they were successively presented with the following scenarios:

- **Task 1**: Login to the application.
- **Task 2**: You are preparing for your maturita and want to practice for your Czech exam. Pick a topic that interests you and practice it.
- **Task 3**: You want to study on the go, download some content for offline use.
- **Task 4**: *(Disconnect device from the internet)* Now you are travelling without internet, practice for your exams.

- **Task 5**: *(Internet reconnected)* Your friend Jane Doe is also preparing for her exams with Hrave, try to connect with her in the app.
- **Task 6**: *(Friend request is generated from another device.)* Your other friend John told you he has tried connecting with you, accept his friend request.
- **Task 7**: See who has been studying the least among your friends in the Czech module.

After the test there was another survey:

1. Did you understand the scenarios and did they feel realistic?
2. Did you get enough feedback about your progress in a lesson?
3. How easy was it to navigate inside the app? Did you ever feel lost?
4. Did something surprise you? Did something feel unexpected or confusing?

4.2.1 Results

Together six people participated in this testing. They were between 23 to 26 years old and some of them use apps very rarely, others use them on a daily basis. Devices that were used were: Samsung Galaxy A3 twice, Samsung Galaxy Core2, Gigabyte GSmart Roma R2 twice, and Samsung Galaxy Note 10.1.

- **Task 1**

  Expected solution The user taps the login button, enters credentials, and confirms.

  User solution Users had no problem logging in.

  Remarks There is a slight problem when the input area is active, any click outside it only deactivates the input field. Therefore, when users clicked on the login button while still having the software keyboard out, the button was not pressed. Users noticed it and pressed it again. This should be fixed.

- **Task 2**

  Expected solution The user taps the button for the Czech module, goes through the sections and starts a lesson, goes through the lesson content, answers questions and finishes with an evaluation dialog.

User solution Users had no trouble finding a lesson. All of them picked and finished a lesson.

Remarks Some users seemed unsure about the continue button, it wasn't clear to them where it would lead them. After trying it once, they had no more trouble. Four users had trouble pressing the radio buttons on their devices, therefore they should have a bigger touchable area.

- **Task 3**

Expected solution The user navigates through the sections, picks a chapter, taps the download button.

User solution All users had trouble with this task. One did not manage to find the download button at all. Two others had searched almost everywhere before finding the button. The other three had found it immediately, however seemed confused and unsure what happened, because the lesson downloaded very quickly and did not show any progress bar or info message.

Remarks The download button should be more visible or there should be a tutorial when it is first shown. Also after a successful download there should be an info popup. One user kept tapping the logo in the main menu, expecting some hidden options to appear.

- **Task 4**

Expected solution The user navigates through the sections that are cached and opens a lesson that was previously downloaded.

User solution All users successfully finished this task. However two of them remarked that they would expect a list of downloaded chapters to be available somewhere in the app.

Remarks Offline content management should be implemented where users could find a list of all downloaded chapters, how much space they occupy on the disk, and the option to remove them.

- **Task 5**

Expected solution The user enters a module, taps the friends button, taps 'find friends' button, enters 'Jane' or 'Jane Doe' or something similar, taps search, taps 'add' button next to the found user.

User solution All users completed this task, however five out of six went first to the profile screen and expected to see friends management options there.

Remarks All of the users wrote the whole name, which signals that it was not clear that they can search only part of the name. Perhaps, after five or more letters the app could show the results immediately and update them with each new letter.

- **Task 6**

  Expected solution The user enters a module, taps the friends button, taps 'requests' button, taps 'add' next to the only item in the list.

  User solution All of the users completed this task, however, because the app only updates friend requests when the main screen or module screen is opened, sometimes the users could not see the request and had to go to the main screen and back.

  Remarks All users understood the buttons accept and refuse (see figure C.3c). The majority of users looked for some friend options in the profile screen, therefore they should be added there.

- **Task 7**

  Expected solution The user sees the leaderboards or opens them with the friends button and announces the name with the least XPs.

  User solution There were no issues here, although one participant did not initially know what XP stands for.

4.2.2 Summary

All participants said they understood the scenarios and found them realistic. They were able to practice in a lesson and understood the feedback, although one reported that they expected to see a wrong answer in red, not grey. The core functionality of the application therefore works well. The main problem was downloading chapters for offline use. Participants said that they did not understand that the lessons would not work offline and had trouble locating the download button. The majority of them looked for friends in the profile screen.

The participants had no trouble navigating inside the app, they naturally used both the hardware back button and the back button in the header. They understood the navigation in the lesson tree and generally felt positive about the application.

In conclusion, the testing provided valuable feedback and revealed some flaws that must be addressed in the future, most critically the offline functionality should be reworked and friend management should be accessible from the profile screen.

Chapter 5

Deployment

The state final high school examination (maturita) was scheduled for the beggining of May 2016, therefore it was important to deploy the application as soon as possible to still be relevant to the majority of the users. It was decided to publish it on Google Play Store because the AppStore review process can take more than a week and because sounds were not yet implemented on iOS. The 'BETA' suffix was added to the name of the app to indicate that not all features of the web are implemented, there might be rendering issues or other problems not found during testing. It was published on April 25th and a link was included in a newsletter email to the users.

Around 80 people downloaded the app and two rated it with four and five stars respectively. This probably means we deployed the application too late for this year's maturita exam, so we should get the application ready for next year, finish all features and polish the UX.

Conclusion

The goal of this thesis was to create an app for the Hrave e-learning platform.

I analyzed the existing web application and worked with the Educasoft company to specify the requirements for the application. I researched similar mobile solutions and designed the application. I assessed the implementation platforms and chose a multiplatform solution. I implemented the application in a maintainable, extendable fashion, and I performed a usability test to evaluate the implemented system. Lastly, the app was deployed on Google Play Store.

Based on Toggl reports, I spent 247 hours on design and implementation of the Hrave app. This does not include the time spent on analysis, research, and writing this text. Petr Miloš, the graphic designer, reported spending 70 hours preparing and discussing the graphic design.

This thesis shows that developing a mobile application with a multiplatform framework is possible, although it must be done carefully to meet user expectations on all platforms. Adobe AIR and Feathers SDK delivered solid performance on tested devices and the related development tools made it possible to develop the app for both platforms in a short time by a single developer. This saves a lot of resources compared to native development.

During this work, I gained valuable insight into the workings of a small software company, their decision making and processes. I also learned more about mobile app design and usability and tried new approaches of developing mobile applications.

I believe this work will be a valuable asset to the Educasoft company and will be improved and built-upon in the future.

Future Work

Apart from releasing the app on iOS AppStore, there are several features that need to be implemented. Firstly, the issues found with usability testing should be addressed and also the effort should lie in fixing all lessons that are

not rendered properly (like unsupported SVG files). Secondly, the Facebook integration that allows users to quickly sign-in or register a new account should be implemented.

The application is ready for multiple domains, however it is branded with the Hrave logo and hrave colors. In the future there could be a configuration associated with each domain, specifying the brand logo and colors.

When a new user opens the app for the first time they are greeted by the login and register buttons. For the future, the user should be greeted, shown a short tutorial, and maybe asked to set a goal. This process is often called 'onboarding' and prevents a percentage of users from leaving the app before they have even seen what it does[57]. If the user sets a goal, for example to study at least 30 minutes every day, there could be notifications to remind the user to stick to this goal.

Bibliography

[1] David, F.; Abreu, R. Information technology in education: Recent developments in higher education. In *2014 9th Iberian Conference on Information Systems and Technologies (CISTI)*, June 2014, ISSN 2166-0727, pp. 1–6, doi:10.1109/CISTI.2014.6876950.

[2] Ministerstvo školství, m. a. t. *Nejčastěji kladené dotazy* [online]. 2013, [cit. 2016-24-04]. Available from: `http://www.msmt.cz/vzdelavani/skolstvi-v-cr/statni-maturita/nejcasteji-pokladane-otazky`

[3] Scio. *Informace o přijímacích zkouškách* [online]. 2016, [cit. 2016-24-04]. Available from: `https://www.scio.cz/prijimaci-zkousky-na-ss/informace-o-prijimacich-zkouskach`

[4] Mediaguru. *Přístupy na web z mobilů a tabletů v Česku stále rostou* [online]. Apr. 2016, [cit. 2016-24-04]. Available from: `http://www.mediaguru.cz/2016/04/pristupy-na-web-z-mobilu-a-tabletu-v-cesku-stale-rostou/`

[5] TechCrunch. *Consumers Spend 85% Of Time On Smartphones In Apps, But Only 5 Apps See Heavy Use* [online]. June 2015, [cit. 2016-24-04]. Available from: `http://techcrunch.com/2015/06/22/consumers-spend-85-of-time-on-smartphones-in-apps-but-only-5-apps-see-heavy-use/`

[6] Ephox. *TinyMCE* [online]. 2016, [cit. 2016-26-04]. Available from: `https://www.tinymce.com`

[7] Information Strategy. *Technology of the week: MOOCs, Duolingo and Khan Academy* [online]. Oct. 2015, [cit. 2016-26-04]. Available from: `https://informationstrategyrsm.wordpress.com/2015/10/05/team-31-technology-of-the-week-moocs-duolingo-and-khan-academy/`

[8] Hamari, J.; Koivisto, J.; Sarsa, H. Does Gamification Work? – A Literature Review of Empirical Studies on Gamification. In *System Sciences (HICSS), 2014 47th Hawaii International Conference on*, Jan 2014, pp. 3025–3034, doi:10.1109/HICSS.2014.377.

[9] Amriani, A.; Aji, A. F.; Utomo, A. Y.; et al. An empirical study of gamification impact on e-Learning environment. In *Computer Science and Network Technology (ICCSNT), 2013 3rd International Conference on*, Oct 2013, pp. 265–269, doi:10.1109/ICCSNT.2013.6967110.

[10] IDC. *Smartphone OS Market Share, 2015 Q2* [online]. Aug. 2015, [cit. 2016-24-04]. Available from: `http://www.idc.com/prodserv/smartphone-os-market-share.jsp`

[11] StatCounter. *Top 8 Mobile Operating Systems in Czech Republic from Jan 2015 to Apr 2016* [online]. Mar. 2016, [cit. 2016-24-04]. Available from: `http://goo.gl/dX8R3P`

[12] Info World. *Native apps crushed mobile Web apps – and that's a good thing* [online]. Dec. 2015, [cit. 2016-24-04]. Available from: `http://www.infoworld.com/article/3012146/web-applications/native-apps-have-crushed-web-apps.html`

[13] Developer Library, i. *About the iOS Technologies* [online]. Sept. 2014, [cit. 2016-24-04]. Available from: `https://developer.apple.com/library/ios/documentation/Miscellaneous/Conceptual/iPhoneOSTechOverview/Introduction/Introduction.html`

[14] PhoneGap. *A high-level summary of what PhoneGap is all about.* [online]. 2015, [cit. 2016-24-04]. Available from: `http://phonegap.com/about/`

[15] Renaux, J. *A year using Ionic to build hybrid applications* [online]. 2016, [cit. 2016-01-05]. Available from: `https://www.airpair.com/javascript/posts/a-year-using-ionic-to-build-hybrid-applications`

[16] Warren, C. *Zuckerberg's Biggest Mistake? Betting on HTML5* [online]. Sept. 2012, [cit. 2016-24-04]. Available from: `http://mashable.com/2012/09/11/html5-biggest-mistake/`

[17] Arora, S. *10 Best Hybrid Mobile App UI Frameworks: HTML5, CSS and JS* [online]. Mar. 2016, [cit. 2016-01-05]. Available from: `http://noeticforce.com/best-hybrid-mobile-app-ui-frameworks-html5-js-css`

[18] Native, R. *React Native* [online]. 2016, [cit. 2016-24-04]. Available from: `https://facebook.github.io/react-native/`

[19] Chambers, M.; Dura, D.; Hoyt, K. *AIR for javascript developers pocket guide*. O'Reilly, first edition, 2007, ISBN 9780596515195.

[20] Adobe Systems Inc. *Building Adobe AIR Applications* [online]. Apr. 2013, [cit. 2016-12-04]. Available from: `http://help.adobe.com/en_US/air/build/air_buildingapps.pdf`

[21] Adobe Systems Inc. *Adobe AIR 3, Frequently asked questions* [online]. Mar. 2013, [cit. 2016-12-04]. Available from: `http://www.adobe.com/products/air/faq.html`

[22] Grossman, G.; Huang, E. *ActionScript 3.0 overview* [online]. June 2006, [cit. 2016-12-04]. Available from: `http://www.adobe.com/devnet/actionscript/articles/actionscript3_overview.html`

[23] Scabia, M. *How Stage3D works* [online]. Oct. 2011, [cit. 2016-12-04]. Available from: `http://www.adobe.com/devnet/flashplayer/articles/how-stage3d-works.html`

[24] Imbert, T. *Introducing Starling: Building GPU Accelerated Applications*. O'Reilly, first edition, 2012, ISBN 9781449320911.

[25] Gamua. *Starling - The Cross Platform Game Engine* [online]. Sept. 2014, [cit. 2016-24-04]. Available from: `http://gamua.com/starling/`

[26] Bowler Hat LLC. *Feathers UI* [online]. 2016, [cit. 2016-24-04]. Available from: `http://feathersui.com/`

[27] Ondřej Paška. *Multiplatform Game Development Using Adobe AIR Technology*. Bachelor's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2013.

[28] A., L. *Why and How to Avoid Hamburger Menus* [online]. May 2014, [cit. 2016-01-05]. Available from: `https://lmjabreu.com/post/why-and-how-to-avoid-hamburger-menus/`

[29] Asana. *Asana* [online]. 2016, [cit. 2016-28-04]. Available from: `https://asana.com/`

[30] Toggl. *Toggl* [online]. 2016, [cit. 2016-26-04]. Available from: `https://toggl.com`

[31] de Alwis, B.; Sillito, J. Why are software projects moving from centralized to decentralized version control systems? In *Cooperative and Human Aspects on Software Engineering, 2009. CHASE '09. ICSE Workshop on*, May 2009, pp. 36–39, doi:10.1109/CHASE.2009.5071408.

[32] Android Open Source Project. *Layouts - Android Developers* [online]. 2015, [cit. 2016-24-04]. Available from: http://developer.android.com/guide/topics/ui/declaring-layout.html

[33] Android Open Source Project. *Button* [online]. 2016, [cit. 2016-02-05]. Available from: http://developer.android.com/reference/android/widget/Button.html

[34] Wharton, J. *Butter Knife* [online]. 2016, [cit. 2016-02-05]. Available from: http://jakewharton.github.io/butterknife/

[35] Robotlegs. *RobotLegs for ActionScript 3* [online]. 2016, [cit. 2016-28-04]. Available from: http://www.robotlegs.org/

[36] Fowler, M. *Inversion of Control Containers and the Dependency Injection pattern* [online]. Jan. 2004, [cit. 2016-27-04]. Available from: http://martinfowler.com/articles/injection.html

[37] Joel Hooks, S. L. F. *ActionScript Developer's Guide to Robotlegs*. O'Reilly, first edition, 2011, ISBN 978-1-4493-0890-2.

[38] Robotlegs. *Documentation for Robotlegs v1.1.2 - Best Practices* [online]. Sept. 2012, [cit. 2016-27-04]. Available from: https://github.com/robotlegs/robotlegs-framework/wiki/Best-Practices

[39] Android Open Source Project. *Supporting Multiple Screens* [online]. 2016, [cit. 2016-27-04]. Available from: http://developer.android.com/guide/practices/screens_support.html

[40] nVidia. *Improve Batching Using Texture Atlases* [online]. July 2004, [cit. 2016-24-04]. Available from: http://http.download.nvidia.com/developer/NVTextureSuite/Atlas_Tools/Texture_Atlas_Whitepaper.pdf

[41] Groe, R. *AS3 HTML Parser Library* [online]. Apr. 2013, [cit. 2016-24-04]. Available from: https://sourceforge.net/projects/as3htmlparser

[42] WWW Consorcium. *Scalable Vector Graphics (SVG) 1.1 Specification [online]*. [cit. 2011-07-07]. Available from: http://www.w3.org/TR/2003/REC-SVG11-20030114/

[43] Labs.zavoo. *Svgweb* [online]. 2016, [cit. 2016-26-04]. Available from: https://code.google.com/archive/p/svgweb/

[44] Adobe. *ActionScript 3.0 Reference - Graphics* [online]. 2016, [cit. 2016-29-04]. Available from: http://help.adobe.com/en_US/FlashPlatform/reference/actionscript/3/flash/display/Graphics.html

[45] Starling. *Dynamic Textures* [online]. Apr. 2016, [cit. 2016-29-04]. Available from: http://wiki.starling-framework.org/manual/dynamic_textures

[46] Android Open Source Project. *Supported Media Formats* [online]. 2016, [cit. 2016-29-04]. Available from: http://developer.android.com/guide/appendix/media-formats.html

[47] FreshPlanet_Inc. *Air Native Extension for AAC playback* [online]. Sept. 2015, [cit. 2016-29-04]. Available from: https://github.com/freshplanet/ANE-AACPlayer

[48] Weber, N. *Building a native extension for iOS and Android* [online]. Aug. 2012, [cit. 2016-29-04]. Available from: http://www.adobe.com/devnet/air/articles/building-ane-ios-android-pt1.html

[49] Apple Inc. *Audio Queue Services Programming Guide* [online]. Dec. 2013, [cit. 2016-29-04]. Available from: https://developer.apple.com/library/ios/documentation/MusicAudio/Conceptual/AudioQueueProgrammingGuide/Introduction/Introduction.html

[50] Android Open Source Project. *Localizing with Resources* [online]. 2016, [cit. 2016-28-04]. Available from: http://developer.android.com/guide/topics/resources/localization.html

[51] Tretola, R. *Beginning Adobe AIR: building applications for the Adobe integrated runtime*. Wrox, 2008, ISBN 978-0-470-22904-0.

[52] Imbert, T. *Getting started with Adobe Scout* [online]. Dec. 2012, [cit. 2016-04-12]. Available from: http://www.adobe.com/devnet/scout/articles/adobe-scout-getting-started.html

[53] Bowler Hat LLC. *How to use the Feathers TextFieldTextRenderer component* [online]. 2015, [cit. 2016-29-04]. Available from: http://feathersui.com/help/text-field-text-renderer.html

[54] Adobe. *Test and debug a mobile application on a device* [online]. 2013, [cit. 2016-29-04]. Available from: http://help.adobe.com/en_US/flex/mobileapps/WSa8161994b114d624-33657d5912b7ab2d73b-7fe5.html

[55] TestFairy. *TestFairy Makes Mobile App Testing Painless!* [online]. 2016, [cit. 2016-29-04]. Available from: http://testfairy.com/TestFairyOnePager.pdf

[56] Nielson Norman Group. *Turn User Goals into Task Scenarios for Usability Testing* [online]. Jan. 2014, [cit. 2016-04-05]. Available from: https://www.nngroup.com/articles/task-scenarios-usability-testing/

[57] Moatti, S. *Mastering Mobile Design: Focusing vs. Expanding* [online]. Apr. 2016, [cit. 2016-02-05]. Available from: `http://blog.invisionapp.com/mastering-mobile-design/`

Appendix A

Acronyms

ANE	AIR Native Extension
API	Application Interface
AS3	ActionScript 3
B2B	Business to Business
CPU	Central Processing Unit
CSS	Cascading Style Sheets
DPI	Dots per Inch
GPU	Graphic processing unit
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
IDE	Integrated Development Environment
JSON	JavaScript Object Notation
LMS	Learning Management System
MVC	Model-View-Controller
MVCS	Model-View-Controller-Service
OS	Operating System
PPI	Pixels per Inch
SDK	Software Development Kit

SQL	Structured Query Language
SVG	Scalable Vector Graphics
SVN	Apache Subversion
UI	User Interface
UX	User Experience
VCS	Version Control System
VO	Value Object
WYSIWYG	What You See Is What You Get
XML	Extensible Markup Language
XP	Experience Point

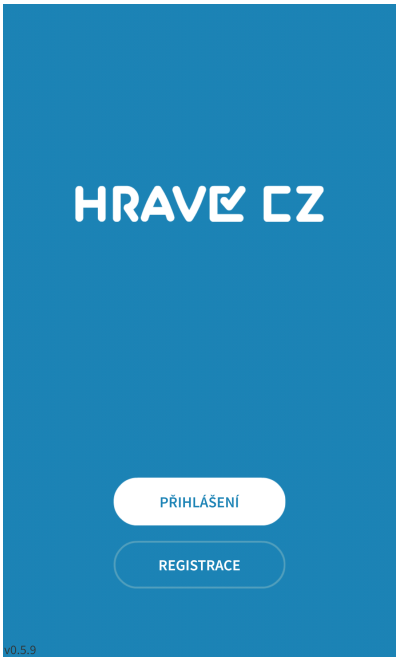
Appendix B

Contents of CD

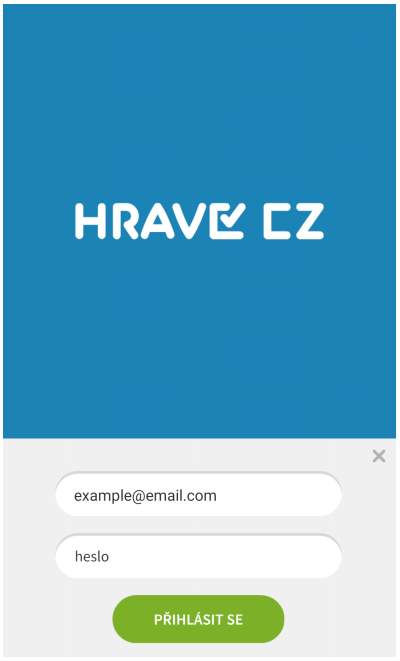
- `readme.txt` ..... the file with CD contents description
- `bin` ..... the directory with executables
- `screenshots` ..... the directory with application screenshots
- `src` ..... the directory of source codes
  - `impl` ..... the directory of source code of the application
  - `thesis` ..... the directory of LaTeX source codes of the thesis
    - `img` ..... the thesis figures directory
- `text` ..... the thesis text directory
  - `thesis.pdf` ..... the Diploma thesis in PDF format

Appendix C

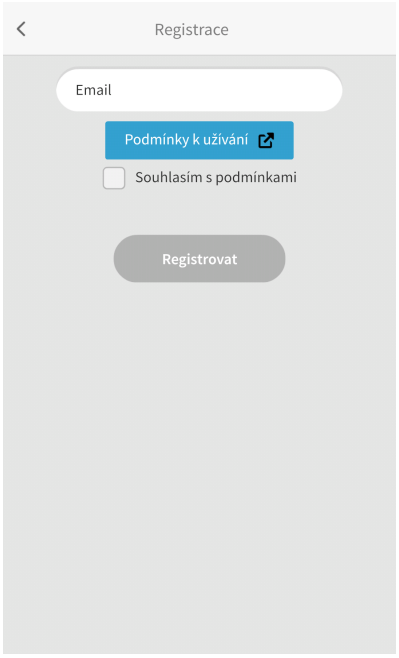
Application Screenshots



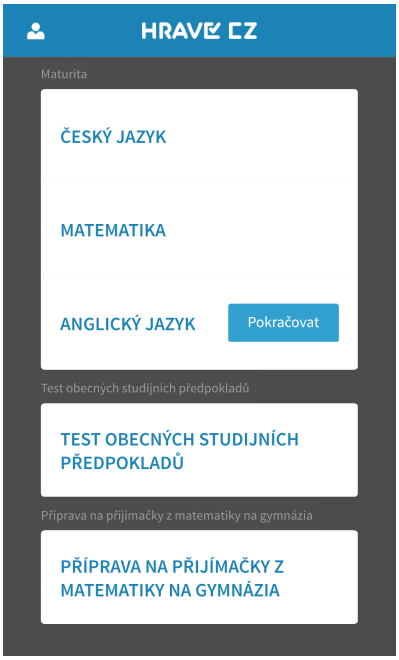
(a) Landing Screen



(b) Login Panel

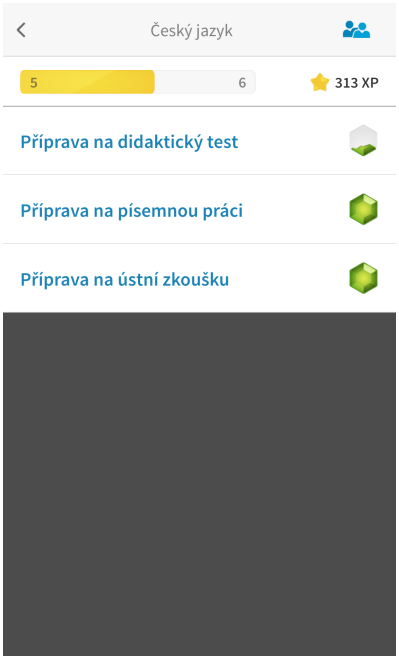


(c) Registration Screen

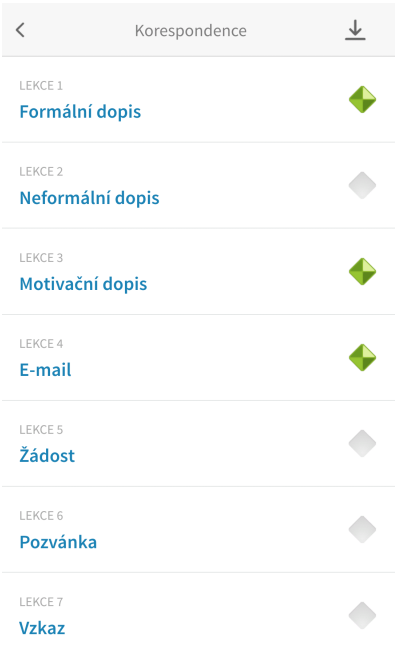


(d) Main Screen

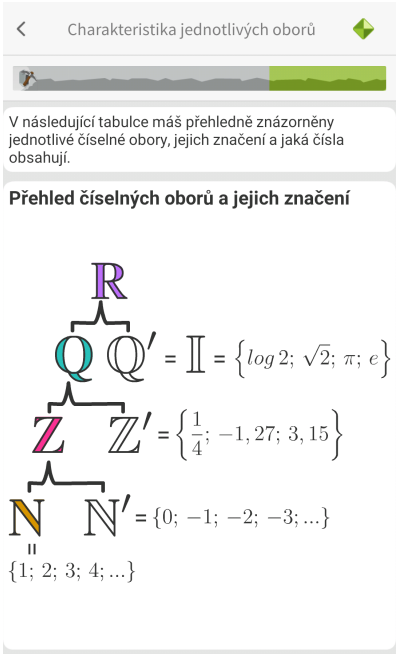
Figure C.1: Application Screenshots I.



(a) Module Screen



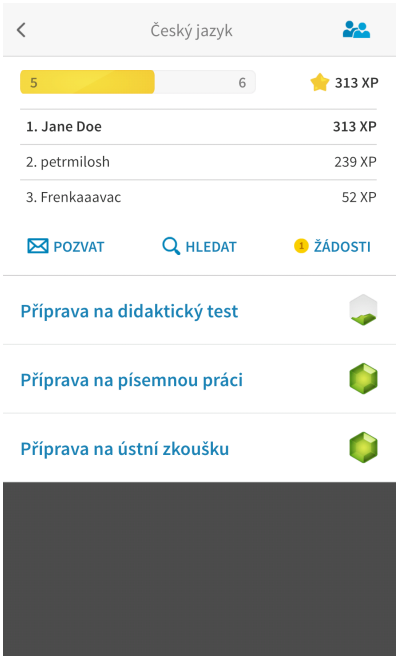
(b) Chapter Screen



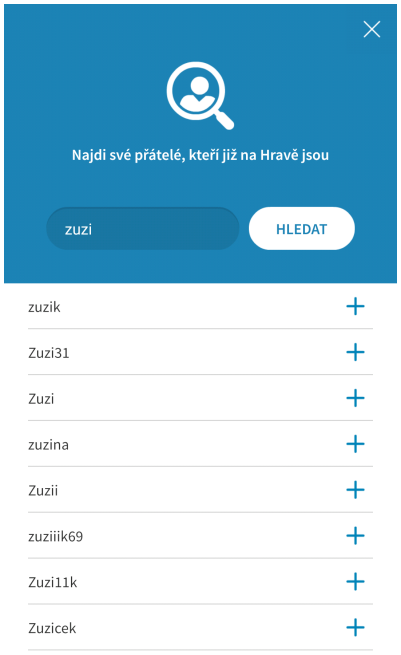
(c) Lesson Content

(d) Lesson Evaluation

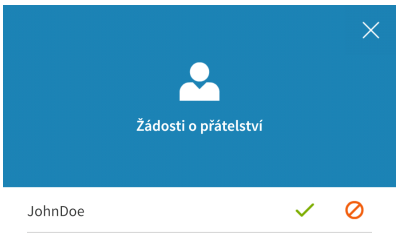
Figure C.2: Application Screenshots II.



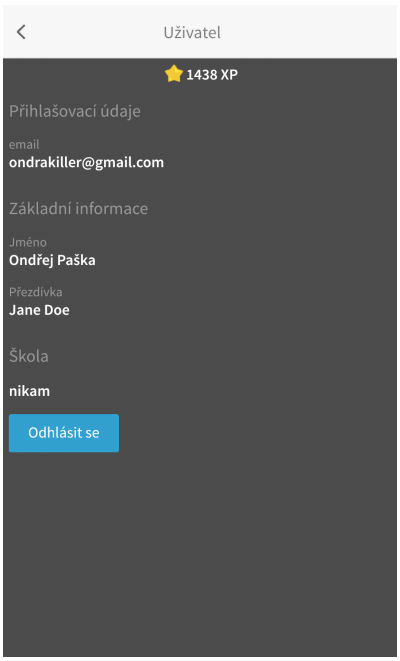
(a) Leaderboards



(b) Chapter Screen



(c) Friend Requests



(d) Profile Screen

Figure C.3: Application Screenshots III.

Appendix D

User Guide

D.1 Installation

The installation of this application depends on the platform and is standard. On Android it is possible to install the application from the enclosed CD. Transfer the HraveMobile.apk file from the CD to your device, ensure that installation from unknown sources is enabled in device settings and open the file on the device. Alternatively, it is available from Google Play Store at the following address:

play.google.com/store/apps/details?id=air.cz.educasoft.hravemobile

The minimum requirements are Android 4.0 and above and ARMv7/x86 processor. The application does not require any special permissions.

On iOS the distribution of software is limited to the AppStore, where the application should be released in the coming months.

D.2 Registration

If you already have an account from the web Hrave.cz, you can use it directly. Otherwise, click on the register button in the landing screen and fill in your password in the registration screen (figure C.1c). You can read the terms and conditions by clicking the button and you must check the corresponding checkbox before pressing the registration button.

Your progress and achievements will be automatically synchronized between the mobile and web versions of the app.

D.3 Accessing Lessons Offline

You can save lessons for offline use by navigating to a chapter and pressing the download button in the header. If the chapter is already downloaded there

will be a green checkmark icon instead of the download button. When you open the application without an active internet connection, you will only be able to navigate to the chapters you have already downloaded.