Master Thesis

Czech Technical University in Prague

F3

Faculty of Electrical Engineering
Department of Computer Graphics and Interaction

Mobile Application for Memsource Cloud Translation Platform

Vojtěch Novák

Supervisor: Ivo Malý
Field of study: Software Engineering
January 2017

Acknowledgements

Děkuji ČVUT, že mi je tak dobrou *alma mater*.

Declaration

Prohlašuji, že jsem předloženou práci vypracoval samostatně, a že jsem uvedl veškerou použitou literaturu.

V Praze, 8. January 2017

Abstract

Memsource Cloud is an online translation platform that helps individuals as well as large translation agencies to manage their translation projects. Memsource Cloud offers tools for the entire translation workflow from document import to final review. Features as Translation memories and Term Bases ensure translation consistency.

This document describes development of an mobile application which will enable the users of Memsource Cloud to access its features through public APIs while—compared to the current web-based solution—offering the user an experience specifically tailored for a mobile application.

Keywords: mobile application development, Memsource Cloud

Supervisor: Ivo Malý

Abstrakt

Memsource Cloud je online platforma pro překlady, která jak jednotlivcům, tak i velkým překladatelským agenturám pomáhá spravovat jejich překladatelské projekty. Memsource Cloud nabízí sadu nástrojů pokrývajících kompletní průběh práce od importu dokumentu, až po závěrečnou revizi. Funkce jako Paměti překladů a Báze termínů zajišťují konzistenci překladu.

Tato práce popisuje vývoj mobilní aplikace, která uživatelům Memsource Cloud umožní přistupovat k funkcionalitám plaformy skrze její veřejná API a oproti stávajícímu webovému řešení bude mít výhodu vyššího uživatelského komfortu díky tomu, že bude navržena přímo pro mobilní zařízení.

Klíčová slova: mobile application development, Memsource Cloud

Překlad názvu: Mobile Application for Memsource Cloud Translation Platform

Contents

Figures

Tables

Chapter 1

Introduction

Memsource is a Prague-based company that developed an online platform for translation management and analytics called Memsource Cloud [1]. The platform is provided as software as a service (SaaS) and uses a freemium business model. It is used by translation agencies as well as freelancers to administer their projects, the documents they need to translate and provides the Memsource Editor tailored specifically for translator needs, allowing them to keep all important information in one place and increase their productivity. In the document, we will use the names Memsource and Memsource Cloud interchangeably and will make a note in case we need to distinguish between them.

To start using Memsource, user has to sign up and choose one of the offered plans. After completing the registration process, they can start using the project management and translation tools. The management part consists of a web-based interface where the user can administer their translation projects, jobs (translated documents), translation memories and the term bases (these terms will be explained later on), users and other features. Closely related is the Memsource editor which is available both within web browser and as a standalone application for all major platforms. The editor serves as a tool for performing the translation itself. The changes made in either of the editors are synchronized. One of the newer features is the analytics bundle which allows the user to see translation progress, performance and get deeper insight into their business.

Memsource is designed to support three kinds of customer groups: in-

[1]https://www.memsource.com/en/features

dividual translators, translation agencies and translation buyers and offers corresponding versions, sub-editions and services to each of those. The Translator edition is meant for individual translators or freelancers and has only a basic set of features offered free of charge. The edition for Translation Agencies adds more features on top of the freelancer version, notably the possibility to work with users, set up user roles and workflow. This allows project managers within the agency to distribute translation jobs among translators and specify workflow, which will be explained later.

The ultimate editions also support advanced features such as workflow and, more importantly, access to Memsource Cloud API. Lastly, the version for Translation Buyers is intended for corporate customers who need to have various texts translated for their business and through Memsource, they're connected to the translation agencies or freelancers who will do the job for them. The Translation Buyers editions include other features specific to the buyers' needs such as pricing inquiry.

1.1 Working with Memsource Cloud

Each of the target groups has a partially different way of working with the Memsource platform. Here, I will cover a usual workflow of a translation agency. Such agency typically has two main types of employees—project managers and translators (linguists). Project managers communicate with customers and receive documents that need to be translated into one or more target languages from them. For the purpose of managing several documents relating to one customer and topic, the manager can set up a project. Following project creation, the manager uploads the files to Memsource Cloud (which currently supports over 50 file formats). In Memsource, such documents are referred to as translation jobs. Jobs have a plethora of properties bound to them, such as due date, linguist who is assigned the job and different settings related to specific file type preprocessing. It is manager's responsibility to make sure the document is imported correctly.

It is then the translator's job to actually translate the document, using the Memsource web or desktop editor. The editor helps accomplishing this task by allowing preprocessing using machine translation algorithms, through term bases and translation memories and quality assurance features.

An important concept here is segment, which usually consists of a single sentence of the translated document. Translation is done one segment after another. Term base is essentially a database of previously translated segments. When translating files that are similar in their content, for example contracts or official documents or different version of the same document, it often happens that there are partial matches in the translated text. Translation memory can spot these similarities and offer the translator a previously made translation. It can also smartly correct small differences such as dates or names. Term base, similarly, is a database of specific terms (single words) and their translations into multiple languages. It can include additional information such as a definition, subject area or industry and etc. Its job is to assure that a term is kept consistent throughout the translated document. The Quality Assurance tools can be used to check if the document meets criteria such as no trailing spaces, no repeated words in close proximity, correct spelling and more.

While the document is being translated, it can optionally go through multiple stages of processing. This feature is called workflow and it allows to keep multiple versions of the same translation job in a project. Workflow may, for example, consist of translation, editing and proofreading. Therefore, once a segment is confirmed in translation, it is propagated to the next workflow level where other agency employer can continue working on it. Once the document is translated, it is delivered to the customer.

Chapter 2

Motivation and Analysis

Memsource operates on the market of CAT (computer-aided translation) tools since 2010 an it is its best effort to provide modern and innovative solutions for translators based on its SaaS model. This effort is fulfilled by a set of described web and desktop-based applications. The current tools developed by Memsource are designed for use on computers with large screen, i.e. laptops, desktops, or large tablets through web browser (with the exception of desktop editor which can be installed on Windows, OS X and Ubuntu).

The sector of mobile devices, however, remain largely uncovered by Memsource. In today's world, mobile devices play an ever important role, allowing people to access online resources from virtually anywhere and at any time. There are a number of studies that show an increasing presence of mobile devices on the internet and in both our professional and personal lives. Statista offers an overview of Smartphone share of visits to websites in the United States in 2014 and 2015, by industry [**?**, statista] This statistics shows that the shares vary greatly between industry, with technology websites having 11.7% share of visits from mobile, while for media and entertainment, mobile accounts for 36.6%. Comscore goes even further and in its study from 2014 [**?**, **?**, comscore] it claims that more than a half of time spent with digital media (social networks, videos, magazines, etc.) in the U.S. is spent on mobile devices. An interesting blogpost from Google AdWords Vice President [**?**, google:adwords]eads that "more Google searches take place on mobile devices than on computers in 10 countries including the US and Japan.". It clearly follows we have to design our software products to play well with mobile devices and the limitations that are inherent to them.

While Memsource can be accessed from a mobile device through its internet browser, the experience is not designed for such use case. There are a number of options and settings accessible (some of which are not used frequently because they support an advanced functionality) and in some cases, this makes the current UI quite complex. Rendering such UI on a mobile device's browser results in inconsistencies as well as limited usability because of small size of some UI elements even though it is programmed responsively.

Memsource, as a leading translation platform provider wants to be able to provide its most important features accessible to customers who are on the go or do not have a computer at their disposal. For reasons I mentioned previously, the existing solution is not suitable for such needs. Therefore, the goal will be to develop an application specifically tailored for use on mobile devices. This app should contain a subset of the features that are currently available and present them in a user friendly manner, keeping in mind the specifics of development for mobile devices. Also, it should by no means aim to replace the existing solutions but complement them.

In some cases, the tools developed by Memsource need to be complex to support different user needs (for example lots of import options for translation jobs). We do not want to bring all this complexity to the mobile application and thus need to find only the functionality we wish to support.

2.1 Requirements

The requirements for what the application has to support were determined gradually by a discussion with the Memsource developer team and CEO, followed by discussions with people from the developer and support teams who know the customer use scenarios and some of who were Memsource users in their previous employments. The outcome of conducted discussions is described in this section.

The context in which the app will be used is a user needing to access, modify or create essentials translation job and project information while being on the move or without access to a computer.

1. User requirements
   1.1. The app will target handheld devices and a narrow group of users—professional users of Memsource Cloud who understand its features.

1.2. The app has to support the project manager and translator user roles

 1.2.1. App has to consider a scenario when a single Memsource user has multiple user accounts and support multiple users logged into the app.

 1.2.2. App will display content for only one user at a time and there will be a simple way to switch between users (i.e. set the active user).

 1.2.3. The number of added accounts will be limited to a maximum of five.

The situation when a single Memsource user has several accounts is not common, but it is not unheard of (This does happen to users who work for several translation agencies). This will make working with several user accounts easier, without the need to log in and out or use several browser windows as is the case when using a computer. Currently, the API uses a token for authentication and this token is sent together with all requests. It is thus possible to make such requests for multiple users from the same device. Note that the API is available to only some of the Memsource editions.

2. Content requirements

 2.1. The app will load all its content through a public HTTP API which is provided by Memsource. In cases where a new API is needed, it will be developed.

 2.2. The app has to make an effort to keep the number of API requests low so as not to overuse battery.

 2.3. App will keep a local copy of downloaded data and allow a read-only preview in case internet connection is not available.

 2.4. The first post-login screen should render within 1.5 seconds after making the first API request when having access to 3G (or faster) internet connection.

  2.4.1. The limit of 1 second should be held by other app's screens.

It follows that we assume the user will have internet connection available on their device at all times when the app is used to create new content and fetch the up-to-date data.

Compared to the web-based service, the app will only support the features that are the most frequently used, to keep its UI simple and easy to work with. At the same time, it should follow the patterns users know from the web version to avoid confusion. More specifically:

3. Functionality requirements

 3.1. Functionality for PM users

3.1.1. The app has to support logging in for Memsource users whose role is project manager (PM) or translator.

3.1.2. App has to support listing projects based on their status (my projects, all, overdue, in progress). The list of project should contain relevant information (name, customer, due date) for each project.

3.1.3. App has to offer project search using the project name, when the project was created (last 24 hours, last 3 days, last 7 days, last 30 days), the client, the project owner and due date.

3.1.4. PM has to be able to create a new project where they have to able to enter a predefined template from which project can be created, project name, client, domain and subdomain, business unit, source and target languages, due date, status, existing Translation memory, existing Term base and note.

3.1.5. App will support deleting projects.

3.1.6. App will support editing projects to the extent of Memsource Cloud.

3.1.7. The app will allow the PM to open a project and see its details (name, user who created the project, date when created, status, due date, source and target languages and project owner).

3.1.7.1. When a PM opens a project, she will be presented with a list of jobs contained in the project.

3.1.7.2. The job list will contain relevant information for each job—job name, linguist name, due date, target languages and status.

3.1.7.3. Within the opened project, PM has to be able to create (upload) new translation jobs from document providers available on the platform (iCloud on iOS and Google Drive on Android and file system of the device, if possible).

3.1.7.4. Adding jobs to project from an email attachment also has to be supported.

3.1.7.5. When adding a new job, user will be able to enter the target languages, select due date, enter linguists for the job and have the option to notify them of the new job.

3.1.7.6. After a job is uploaded, user has to be notified about it. Similarly, user has to be informed about unsuccessful file uploads and requests.

3.1.7.7. App should only support creating jobs from the most common file types (MS Word, Excel and multilingual Excel, Powerpoint and HTML) and cover their file import options to the same extent as Memsource Cloud.

3.1.7.8. After successful import, user will have the option to be presented with a read-only preview of what the import looks like, to get a feedback on whether the import was successful.

3.1.7.9. User needs to have the ability to select individual job, as well as a shortcut to select all jobs, to download, edit and delete it. Editing consists of changing the linguist, status and due date. Similarly to searching and filtering or projects, the app will allow to filter jobs by name, status, target language, linguist and due date.

3.1.8. The application will not cover support for translating, i.e. Memsource Editor will not be a part of it since we found that it is overly complex to be used on a small screen of a mobile device.

Note: The advanced project creation options for machine translation, analysis and other which are visible in Memsource Cloud, will not be included.

3.2. Functionality for Linguist users

3.2.1. The app will support listing projects that are visible to her, filtered based on status (new, accepted, completed). The list will contain relevant information—name, date created, owner, and source and target languages.

3.2.2. User will be able to open a project, download or preview a job and change its status (accepting or rejecting and marking them as completed).

3.2.3. For the purpose of downloading multiple jobs, app will allow to select particular jobs and provide a shortcut to select them all.

3.2.4. Same as with a PM, translator will have the ability to search based on job filename, status, target languages and due date.

4. Other requirements

4.1. It is important the app be developed for at least the two major mobile phone platforms, that is Android and iOS.

4.2. The app must be developed using programming languages that Memsource developers are already familiar with, that is one or more of: Java, Groovy, C++, Javascript or any other technology if it brings substantial benefits.

4.3. For the user, the app should look and feel as close to a native app as possible. It should be kept simple and support different screen orientations when suitable.

4.4. The application has to support error reporting to allow Memsource continually improve the user experience and app stability.

Summary of requirements

Non-functional requirements

1. app has to run on iOS and Android and be stable
2. app should be written using technologies already used in Memsource
3. UI components should have a platform-native look
4. UI transitions have to feel smooth
5. UI loading times not greater than 1 second on high-speed internet connections
6. UI follows the design guidelines of both platforms
7. app has to be intuitive to Memsource users
8. app should minimize number of API request, to save resources
9. the software has to be maintainable and testable
10. use external software packages only if their license is suited for commercial use

Functional requirements—the app has to:

1. allow a Memsource project manager or translator to log in and out
2. support multiple users logged in at the same time and allow to switch between accounts
3. list projects and allow to search in them
4. list jobs and allow to search in them
5. support adding jobs and projects
6. allow to edit projects and jobs
7. store user credentials in a secure manner and otherwise unaccessible even on rooted Android devices
8. support error reporting
9. use Memsource API to acquire and send the content
10. present its UI in English only

2.2 Prototyping

Based on the collected requirements, a set of mockups was constructed. Some of these were later used to construct a prototype for an Android device that was used for tests with users. Both the mockups and the prototype consider the project manager role because its feature set is a superset of the one of the linguist role.

The mockups were discussed with employees of Memsource support team. Different ideas of presenting information to the user were brought up and consulted. Memsource support members provided feedback and deeper insight on how different features are used and whether they are needed. Understanding the workflow was important to designing the final mockups.

In the end, the prototype largely follows the structure of Memsource cloud, but it is simpler in terms of the number of supported options and gives a platform-specific feel. The following pages contain several figures that present selected mockup screens.

The very initial design can be seen in figure 2.1. Its original idea was to be focused on job listing after user login, because the status of translation jobs is, perhaps, the information a PM wants to get. This view would contain jobs from different projects listed together to give an overview of the jobs that need to be completed soon. This idea, however, was turned down because of mixing different projects together and also because there is no such equivalent view in Memsource Cloud, which could confuse the user.

Figure 2.2 shows the next design of the project listing. There are controls for creating a project, searching, filtering (using the chevron) and getting the important information about user's projects. In figure 2.7 you can see how filtering the projects changed in the final prototype.

Figure 2.3 shows a mockup of an opened project with translation jobs listed (first screen). The second screen shows a job being selected and the consequent changes in the navbar: the user can choose to edit, select all or download job.

Figure 2.4 contains the screens for adding a new job to a project from within the application. The figure shows two states of the same screen, first screen is waiting for user to enter needed information, the second displays

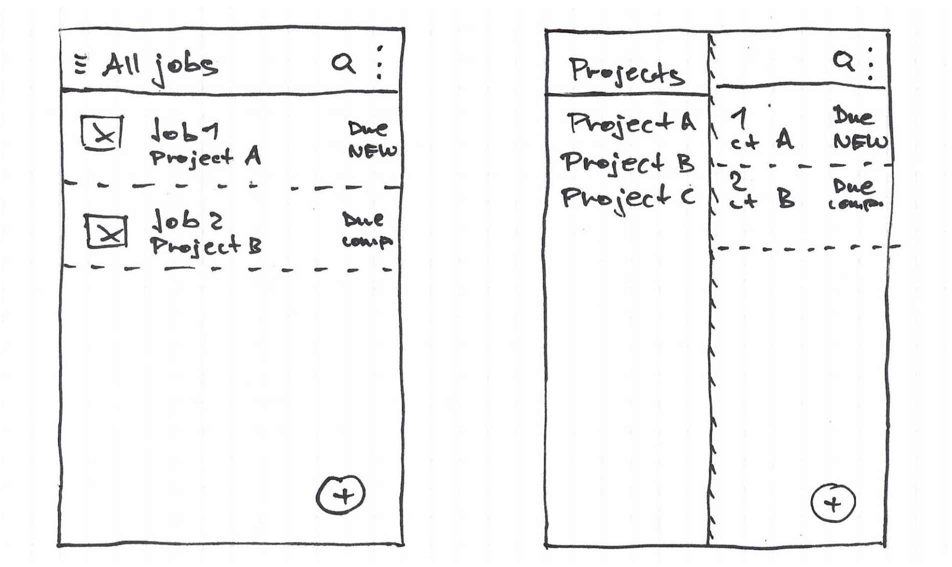


Figure 2.1: Initial post-login screen (left). Project list and opening a project would be accessible from the drawer (right).

the state when the information is entered, along with two files chosen for upload. In the prototype, the layout of these screens was preserved, except for division into two tabs: one for uploaded files and second for import settings.

For iOS, which has different interaction patters compared to Android, I created separate mockups of what the UI could look like. The following figures show the mockups for project handling.

2.2.1 Testing with users

After the mockups were positively received, I created a prototype for Android. The prototype was created using Axure RP 8.0 [1] which is a software for creating different kinds of UI prototypes. There are widget libraries available, which contain ready-to-use Android and iOS UI elements. Figures 2.7, 2.8 and 2.9 show selected prototype screenshots.

To verify the created prototype, I conducted two informal tests with users. Axure provides Axure Share service to share projects, with and Android app [2] available on the Google Play Store, but this app proved not to be suitable for testing because it does not scale the UI well. Instead, I took advantage

[1] http://www.axure.com/
[2] https://play.google.com/store/apps/details?id=com.axure.axshare

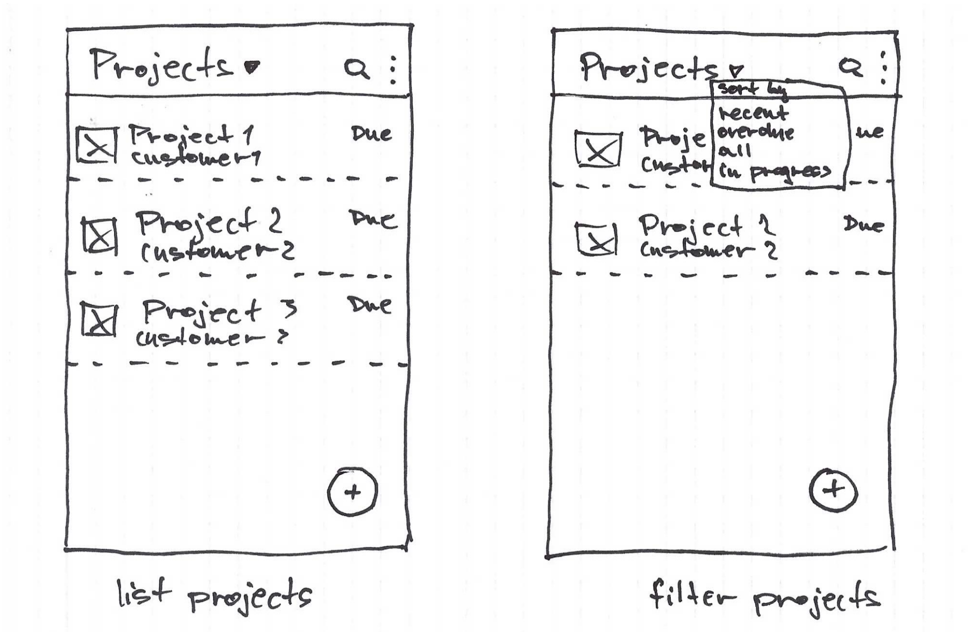


Figure 2.2: The improved mockups showing the project list screens. Second screen shows the chevron active, where user can filter displayed projects.

of Axure's ability to export created project as HTML which can be viewed directly on the device, for which I used the Kiosk Browser [3] app which allows to display content full-screen.

To help keep the users relaxed, I explained the purpose of the application we were about to test and that we were testing only an initial prototype to catch its flaws, and not testing their abilities of working with Android.

The users were given a list of tasks corresponding to a possible walkthrough of the app. The text is following:

You are a Memsource Cloud user and your role is project manager. Log in using the username “user” and password “pass”. View translation jobs in Project 1 and download job whose name is “Job name” onto your device. Then create a new job from the “document.docx” which is available on Google drive. For the job, select English and German as target languages, due date as 2nd January 2016, 11:00 am and enter linguist name. You need the file be imported with comments and hidden text. Then, create a new project as new project name, enter “test project”, select “client 3” as the client and select arbitrary parameters for the other options.

[3] https://play.google.com/store/apps/details?id=it.automated.android.browser.kiosk

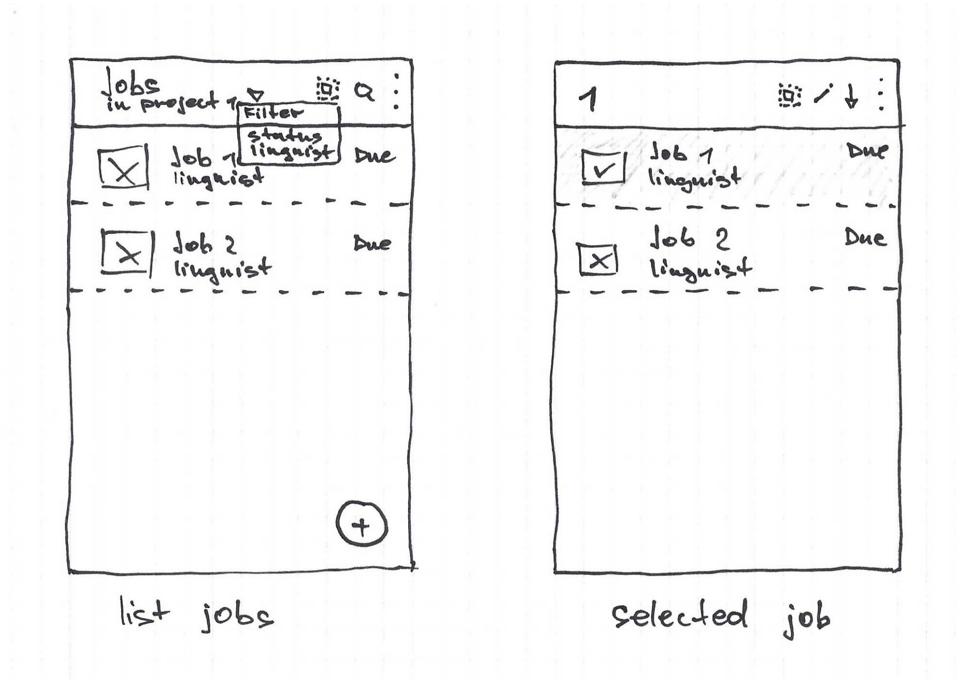


Figure 2.3: Job list (left) and a mockup where "Job 1" is selected and different actions are available for it (right).

The informal testing was conducted in the company offices with two members of Memsource support team who both were owner of a mobile phone running Android. Test was conducted using LG Nexus 5 with the prototype running in the aforementioned Kiosk Browser. The downside of this setup was that the back button of the prototype was not available and in one instance (before adding a new project), this required intervention into the test process and consequent finding that a back button should be added to the navigation bar. Also, the generated html prototype seemed to have issues with entering text into textfields—they were accessible only after a long press instead of a simple tap. I have not found the root cause of this and needed to inform users of this issue before starting the test.

2.2.2 Test results

The test was completed by both users. However, during the test, two mistakes present in the UI were reported (problem with import options and adding new project). Both users complained about unintuitive icons, which was especially true of the white cloud icons in the upper right corner of the screen. After filling in all information for a new project, one user asked if that was the icon they were supposed to tap.

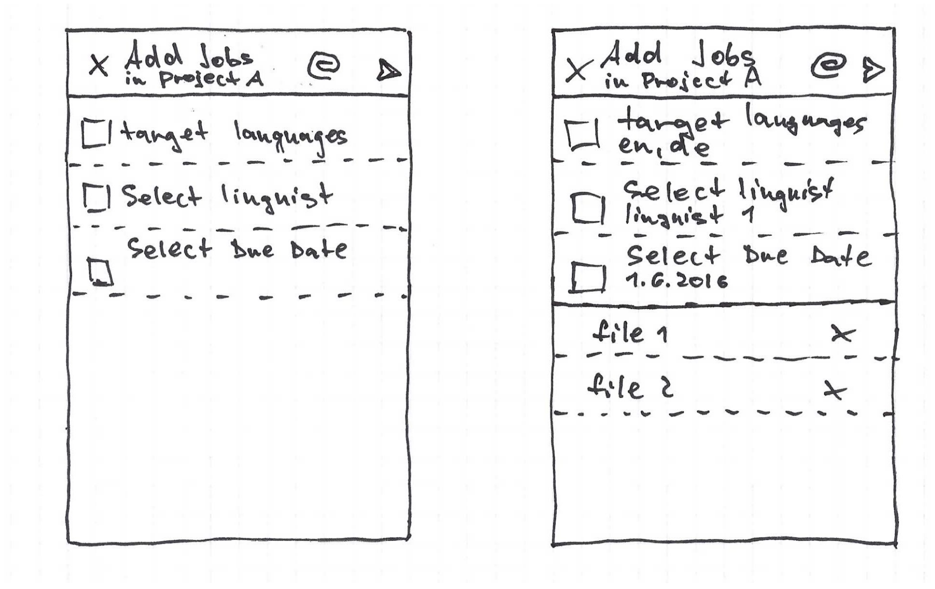


Figure 2.4: Adding a new job to a project.

For the second iteration of testing, I used an improved prototype which fixed the flaws we found in the first iteration. Also, I stopped using the kiosk app and opted for the Axure Share Android app after fixing the scaling issue. The second test was successful and users reported they were satisfied with the experience. One tended to play with the prototype beyond the extend of what it was made for and complained some buttons were not functional. I do not consider that a problem, since this was mainly a horizontal prototype with only a particular interaction path fully implemented.

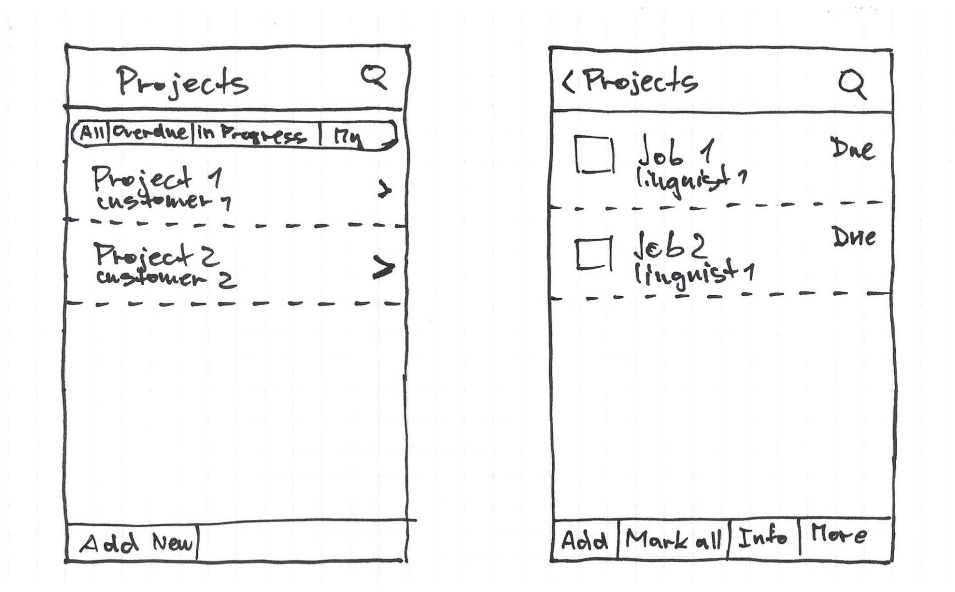


Figure 2.5: Listing projects on iOS and viewing jobs within a project.

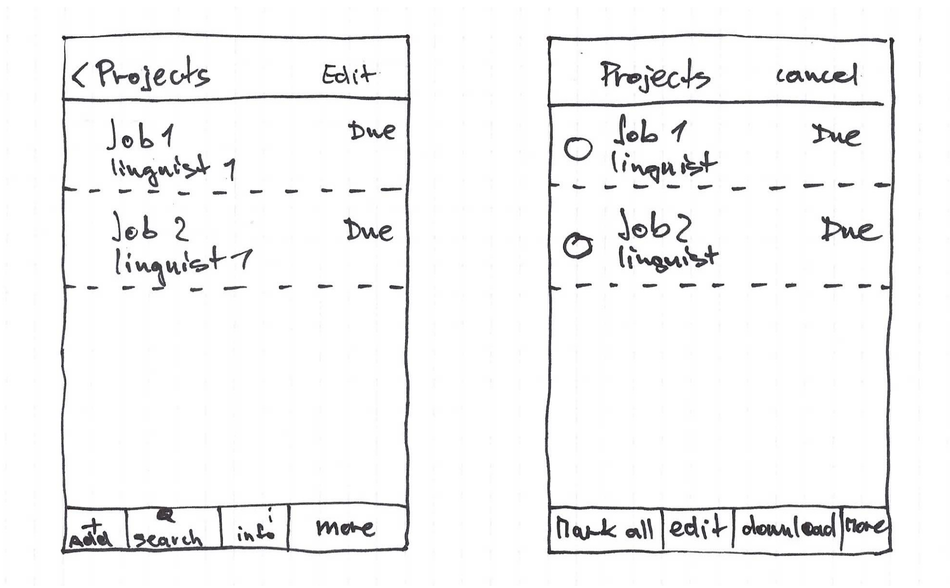
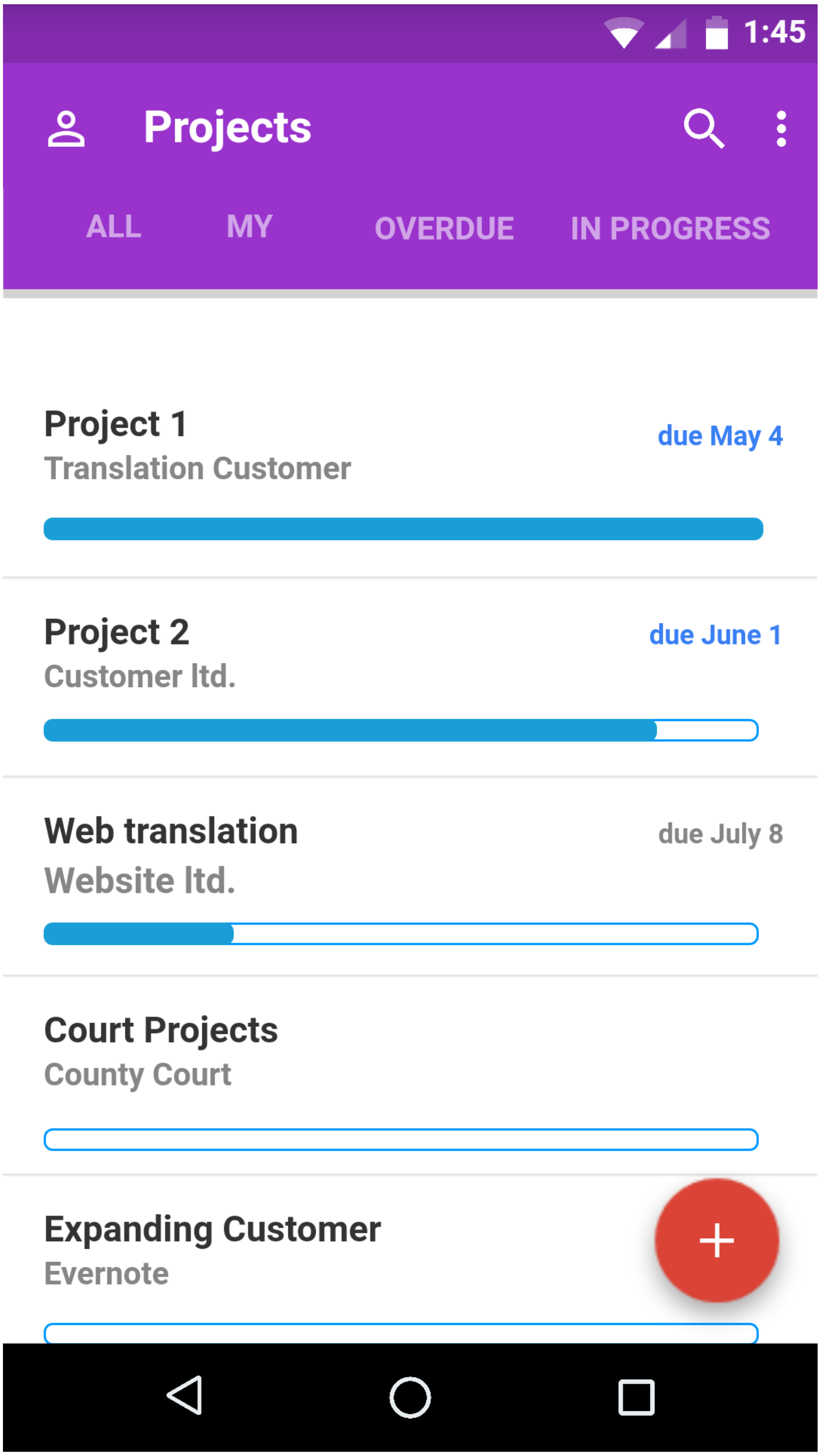


Figure 2.6: Different way of listing jobs within a project on iOS. The second screen displays the state of the first after clicking on the 'edit' button.



Project 1
Translation Customer
due May 4

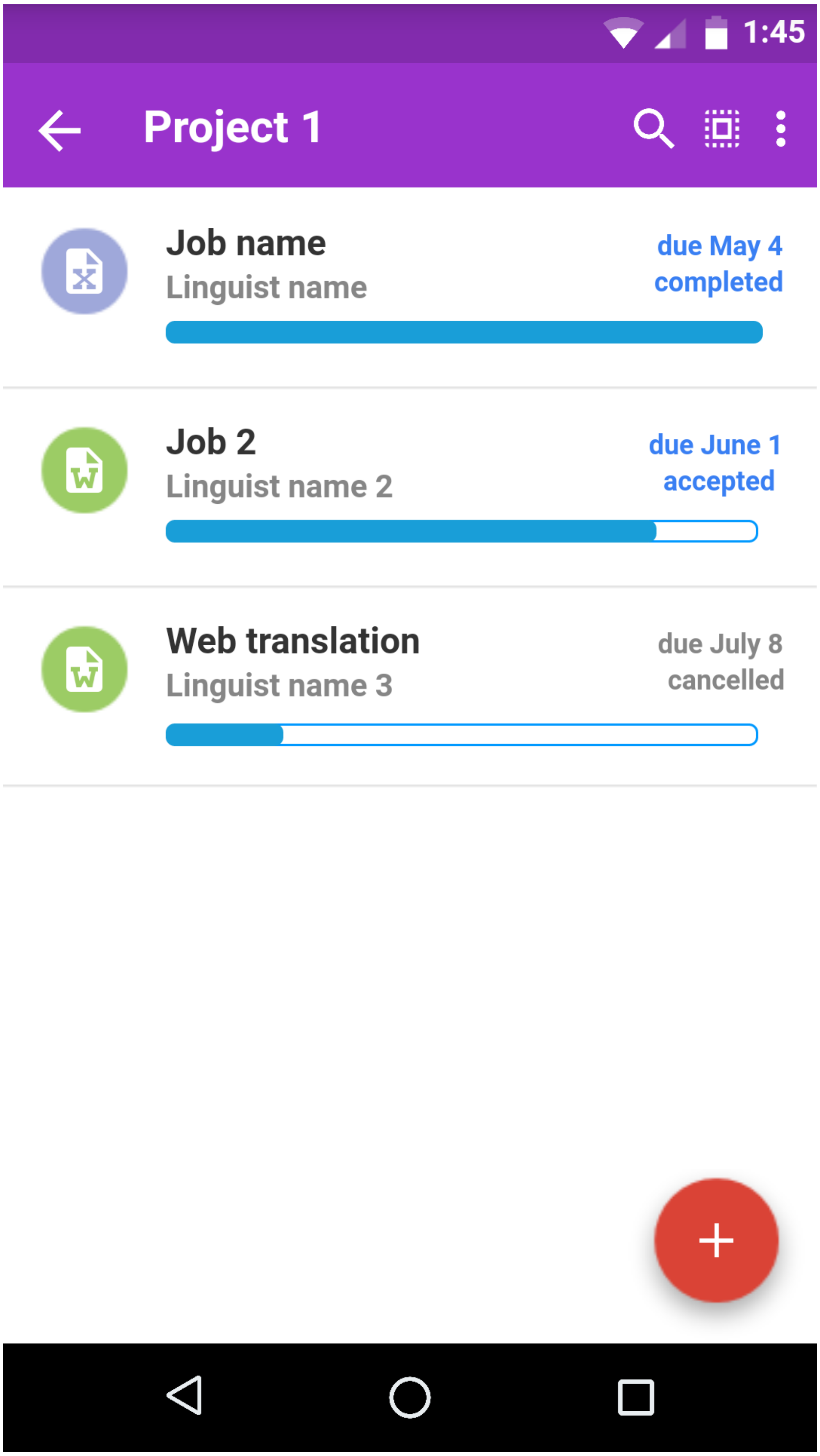
Project 2
Customer ltd.
due June 1

Web translation
Website ltd.
due July 8

Court Projects
County Court

Expanding Customer
Evernote

Figure 2.7: Project listing in the prototype used for testing with users.



Job name
Linguist name
due May 4
completed

Job 2
Linguist name 2
due June 1
accepted

Web translation
Linguist name 3
due July 8
cancelled

Figure 2.8: Listing jobs in a project and creating new job.

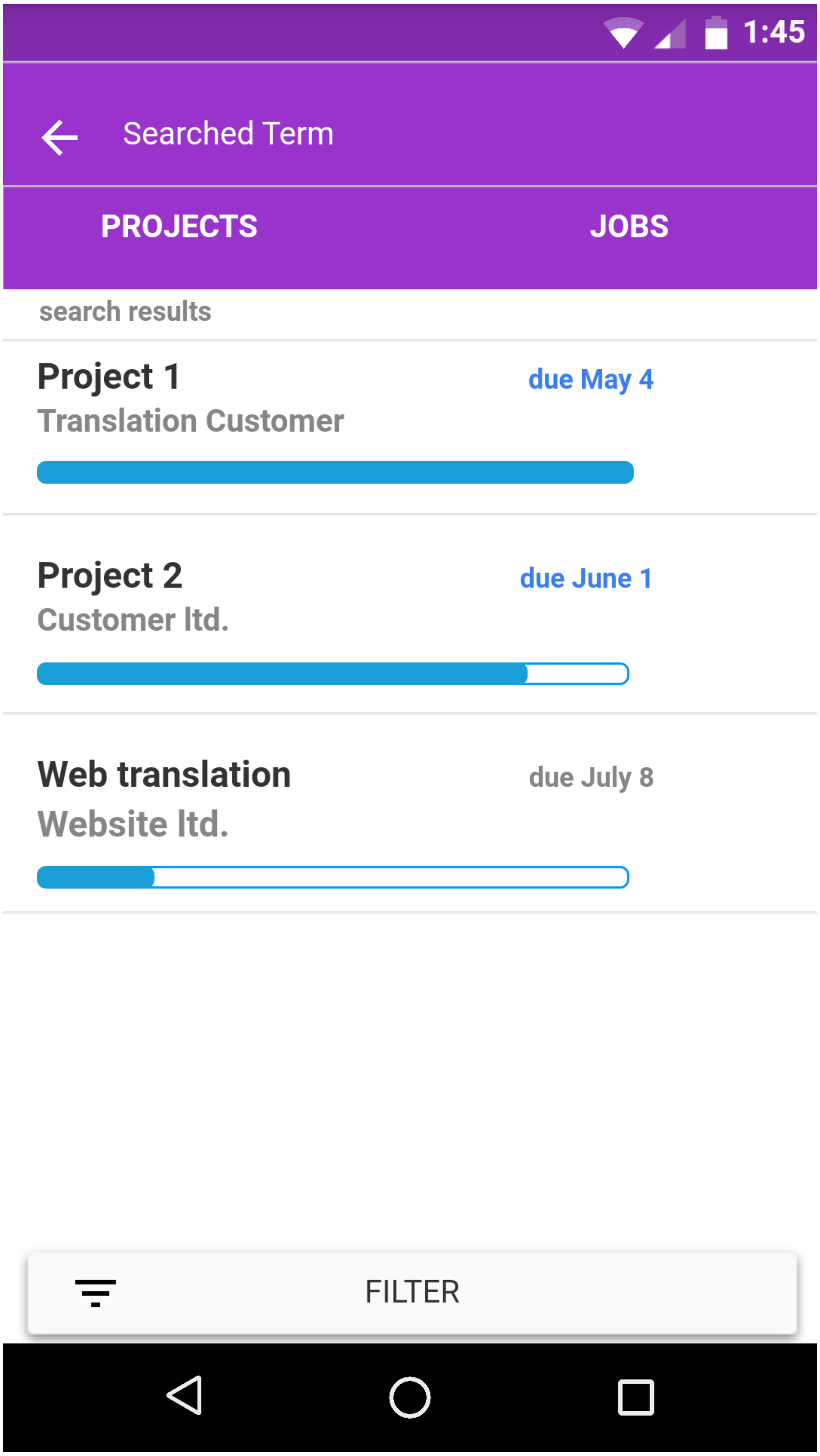


Figure 2.9: Project search by name and a selected job in a job list.

Chapter 3

Analysis of Platforms and Development Tools

Today's market of mobile devices is largely divided between two major platforms—iOS and Android. With 80.7% for Android and 17.7% for iOS, these two alone made up more than 98% of worldwide sales in 4Q15, according to Gartner [?, gartner] Windows Phone comes third with 1.1% of sales. Interestingly, in its March 2016 report, Kantar Worldpanel shows the sales of vary greatly among different states [?, kantar] For example, while iOS has a sales share of 56% in Japan, it only has 17.8% in Germany. As of Q1 2016, Android is growing in Europe, at the expense of iOS and Windows Phone, whose sales have been dropping in the UK and France where Windows Phone was historically relatively successful.

3.1 Xamarin

Xamarin is a framework for developing native apps for iOS, Android and Windows Phone. The company was founded in 2011 and was acquired by Microsoft in 2016. All of Xamarin is now open-sourced on GitHub [1]. Xamarin divides into four main parts: Xamarin.iOS, Xamarin.Android, Xamarin.Windows and Xamarin.Forms. The first three offer access to native APIs for the particular platforms, where Xamarin stresses that "Anything you can do in Objective-C, Swift, or Java you can do in C# with Xamarin"

[1]https://github.com/xamarin

[?, xamarin:quote] When using these, the developed solution consists of one project per supported platform (which contains the platform-specific code — mainly the UI) plus single project whose code is shared. Reportedly, this approach can result into about 75% code reuse [?, xamarin:codereuse] Xamarin.Forms is an attempt to bring code reuse event higher, usually more than 90% [?, xamarin:codereuse]y sharing also the UI code.

As explained, UI in Xamarin can be either defined specifically for each platform within the platform's application project or cross-platform using Xamarin.Forms [?, xamarin:forms]

Taking the first way is recommended for apps with interactions that require native behavior, apps which use many platform-specific APIs or cases when custom-tailored UI has higher priority than code sharing. With this approach, each app will have its own UI defined in C# code or XAML (an XML-like sytax for UI description) which can be done in a graphical UI designer.

Using Xamarin.Forms is best for apps that require little platform-specific functionality and apps where code sharing is more important than custom UI. Xamarin.Forms UI can be done either in C# or in XAML but without the support of UI designer.

Development with Xamarin is done under Windows or OSX and the language of the framework is C#. The code must be built and sent onto a device or emulator, which can take a considerable amount of time. Debugging is done in Visual Studio. Xamarin allows to write the code shared by all platforms in form of a Shared Project or a Portable Class Library [xam16] which are described in the following sections.

3.1.1 Shared Project

Shared Project is the simplest way to share source code between platforms. This way, a cross-platform app that supports Android, iOS and Windows Phone would require an application project for each platform. Additionally, there would be a Shared Project for the code common to all projects.

The code within a Shared Project can be branched into platform-specific parts using compiler directives (i.e. using #if ___ANDROID___). The application projects can also include platform-specific references that the shared code can utilize. The downside to this approach is that a Shared

Project has no output assembly. During compilation, the files are treated as part of the referencing project and compiled into that project's DLL. This does not allow to distribute the code from Shared Project as an independent library.

3.1.2 Portable Class Library

Portable Class Library addresses the fact that Shared Project cannot be distributed as a standalone library. Portable Class Library offers the possibility to distribute it independently of the mobile app.

The disadvantages are that it is not possible to use compiler directives to reference platform-specific features and the fact that different platforms often use a different subset of the .NET Base Class Library (BCL) and therefore only such subset is available to use. This, however, can be to some extent circumvented by the Provider pattern or Dependency Injection. That way, the actual implementation is coded in the platform projects against an interface defined within the Portable Class Library.

3.2 React Native

React Native (RN) is a counterpart of the popular web development framework React and is also developed by Facebook which uses in several production apps and "will continue to invest in it" [2]. It was first released in 2015, which makes it the youngest among the covered frameworks. React is popularized under the slogan "Learn once, write anywhere.", as opposed to e.g. Java whose goal is that one codebase runs anywhere, this means that once a developer learns React, she can use her skills to write apps on multiple platforms (web, Android, iOS, etc.) using just this one tool.

React originally started as a tool for describing user interfaces for the web, and rapidly became popular within the web development community. However, it was quickly recognized that React's usage was not limited only to web.

React describes the user interface through reusable components which tell

[2]https://facebook.github.io/react-native

what the UI is supposed to look like. It is then the matter of transforming the description into a user-facing UI. On the web, this is the task of React-DOM which uses the Virtual DOM tree as a layer of abstraction between developer's code and what is rendered in the browser. When programming mobile apps, this abstraction is handled by React Native.

Indeed, React Native takes a different approach from traditional native and hybrid development environments. RN application code is written in JavaScript which communicates with a JavaScript runtime (JavaScriptCore engine) running on the device [rn:16b]. This JavaScript runtime is in turn responsible for bridging the calls onto the native platform APIs and back. This way, user can access any native functionality and get information back in a callback or promise payload.

When developing with RN, the developer creates or makes use of ready-made UI components and uses them to compose the application UI. The components are written in a combination of JavaScript and XML tags, called JSX. Optionally, Flow [3], a static type checker for JavaScript can be used. Also supported although not so frequently used are languages trat transpile to Javascript, such as TypeScript [4].

Important features of RN are its live and hot reloading [rn:16a]. Live reloading enables the developer to apply the code changes to the app running on a device or in an emulator quickly. Live reload in fact takes about five seconds on our development machine. This is a tremendous improvement over traditional native development, which until recently—with the introduction of Instant Run into Android Studio—suffered from the slow process of building an app and loading it into a device or emulator.

The other feature called Hot Reloading offers essentially the same functionality as Live Reloading with the advantage of being faster and preserving the application state—the screen displayed before and after the Hot Reload is the same—thanks to which the developer does not need to navigate through the app to the screen where the change is being done. This is especially helpful for making changes to the UI layout and styles because Hot Reload needs about one second to take effect. These features can significantly accelerate app development and improve the developer experience. The downside of Hot Reloading is that it mostly works for simple changes in the UI but fails when modifications are of larger extent.

Debugging RN is accomplished through running the code in Chrome browser

[3]http://flowtype.org/

[4]https://www.typescriptlang.org/

or external debugging tools (such as those included with Visual Studio Code or Webstorm) where user can set breakpoints and work with code similarly to working on web development. In this case, all the JavaScript code runs within Chrome's V8 engine itself and communicates with native code via a WebSocket.

3.2.1 Native Modules

When a developer needs functionality which is not already provided by RN, they will often find such functionality already implemented by the large community which surrounds RN. In such case, the component is available through the Node package manager (npm).

In case the functionality is not yet implemented, the developer can create a native module [rn:16c] for it. Native module consists of code written in Java (for Android) and Objective-C (or Swift) for iOS which implements the desired functionality and of JavaScript code that will serve as a 'glue' between the native code and the app's JavaScript code. Native modules give developer the freedom to implement any functionality desired, as long as it is available on the platform, but have the downside of needing to code both for iOS and Android. The native code is then invoked from Javascript through the RN bridge and results (if any) can be passed back by a Promise or callback.

3.3 Ionic

Currently in version 2 Ionic [5] is another successful framework for developing cross-platform mobile apps which was initially released in 2013. Ionic is a hybrid framework, meaning an app created with it runs in a WebView, same as a website would—with the important difference that it can also use the native device APIs. It supports iOS, Android and Windows Phone.

Just like the aforementioned frameworks, Ionic is open source and offers a set of mobile-optimized components written in HTML, CSS and JavaScript. Ionic 2 is integrates with Angular 2, a framework for web development from Google. Ionic has put o lot of work into providing components that are styled according to each supported platform, thus saving the developer's time by

[5]http://ionicframework.com

not having to spend valuable time by styling the UI. Compared to React Native and Xamarin, Ionic gives less flexibility in customizing the app for different platform. With Xamarin and RN, developer can make the app look quite different (at the expense of writing more code) while this is limited on Ionic. Depending on the particular app context, this can be both a downside or a benefit.

Ionic 2 developer can optionally choose to develop in TypeScript, which is a language that compiles to plain JavaScript. As its name reveals, the most important TypeScript's feature is addition of types to JavaScript. This can reveal errors before they happen, and gives extra information to both the developer and IDE (Integrated Development Environment) which therefore can offer better code completion. TypeScript is basically a competitor of Flow.

Ionic runs inside Apache Cordova, a mobile application development framework which provides access to native platform features such as camera, sensors, filesystem or contacts. Access to arbitrary features can be allowed through plugins, which are composed of a single JavaScript interface used on all platforms, and platform-specific code code which is called from JavaScript. In this respect, Cordova Plugins are similar to native modules in RN.

Ionic is about more than only mobile app development. Ionic offers features like ionic lab, which allows to run iOS and Android apps one next to the other in browser as well as in a GUI application for Windows and Mac. There is also the Ionic Market, which contains lots of starter templates and themes. Ionic's View app allows developer to easily share apps with customers and testers. Ionic Creator is an prototyping tool where developer can drag and drop components to create a simple app an even export it as an Ionic app.

Also, with live reloading, the iteration process is a lot faster than traditional procedures involving compilation.

3.4 Conclusions

There is currently a very strong competition in the area of multiplatform mobile app frameworks and choosing the right one for one's needs is difficult. All of the researched solutions are very capable. After developing simple proof-of-concept apps using the three described frameworks I first ruled out Xamarin. Although C# can be considered a very mature and powerful language, the reasons for ruling Xamarin out were the need to write UI twice (we did not want to use Xamarin.Forms because we were uncertain about whether it would not limit us and the available demonstrations apps written using Xamarin.Forms were not stable) and its slow development iteration cycle which for me—as someone having experience with web development—felt like a big drawback.

Ionic 2 is a popular framework. After creating simple application in it, I could not help but notice very slow startup times: between 6 and 9 seconds for a very simple app. This issue was confirmed by posts in the community forum and according to an Ionic representative, the startup time will be improved. Moreover, after installing several apps developed in Ionic we noticed not all of them work on all devices (probably due to cpu family), which left me with mixed feelings. The advantage of Ionic 2 surely layes in strong community and lots of readymade UI widgets.

React Native was chosen after difficult comparisons. Its advantages are that it is backed and used by Facebook, it is being developed at a quick pace and has a big community with lots of components available or in development. It also provides greater flexibility than Ionic, and the ability to communicate with native code which is at the core of RN is a need for our use case since the app will have to deal with background file uploads. Obvious disadvantage is its immaturity and probably the need to work more to get eye-appealing designs.

Chapter 4

Design

Design here

4.1 Issues

Issues here

4.1.1 Navigation

React Native started off with two solutions for navigation (i.e. For letting the user transition between different screens of the app) - the Navigator and navigatorIOS. Navigator is implemented entirely in Javascript, runs on both platforms and tries to mimic the appearance of native ios navigation, while NavigatorIOS leverages the native navigation of iOS. They originally started as two competing implementations solving the same problem (todo ref) with the goal of assessing which of the two solutions should be supported further on. Ultimately, the Navigator solution was found to be better for reasons described later on and facebook used it in the F8 and Facebook ads apps.

There is one notable drawback to Navigator - it is only trying to mimic the native navigation. This includes navigation bar with its animations and transitions, as well as transitions to and from different pages of the app and implementation of the swipe back gesture. This, to a certain extent, can make an app which uses Navigator feel less native. Having navigation controlled by Javascript has its benefits - most importantly it allows for complete control of the navigation state, while with navigatorIOS, some information sent from Javascript to the native side gets stored in the native code where it is managed by ios internally which makes it hard to keep the information in Javascript up to date. The difficulty of keeping Javascript in sync with ios was the main reason why Facebook decided to favor the Navigator.

Another potential benefit of using Navigator is the fact that it can be used on Android as well (this is thanks to the fact it is written in Javascript). This, is not of a great importance because it would make an android app look like an ios app but it allows for reusing the same code to transition from one page to another, with the platform - specific details handled by configuration.

Wix Some would say that not using the native navigation breaks the promise of React Native - that is to be able to create apps that are indistinguishable from native ones. This is the reason why Wix (an Israeli mobile and web development company) decided to implement a native navigation for react native, called react-native-navigation. At the time of writing, there is not a stable release of this package available.

Navigator and navigatorIOS are two initial implementations which both attempted to bring the navigation from ios to react native. Over the course of time new issues with using Navigator emerged and there was a need to come with new way of managing navigation state. At the time of writing, this new solution is called NavigationExperimental. It offers improved api and support for several navigation stacks at once.

4.1.2 Platform-specific look and feel

React Native does not aim to provide developers with a way to run the same code on both platforms, instead it takes advantage of the "learn once, write anywhere" paradigm and allows her to create apps for both platforms while writing code using the same syntax.

Due to the nature of how both platforms are interacted with, we need to have ability to make the user experience different per platform. As an example, take the datepicker on Android versus the ios picker, or the Android toolbar which often offers several actions (some with icons) versus ios toolbar which usually contains the title and no more than 2 or 3 actions. Actions that on Android would be included in the toolbar, are often presented in a bottom bar on ios, or hidden in an action sheet. Also note how android works with long presses for item selection, while this is usually done by an edit button in ios nav bar This requires us to write separate code that would take the actions and display them differently on each platform.

React Native offers two ways how to go about this. First is through the platform module which gives information on the platform and its version. Another method is to use different file extensions (ie. android.js or ios.js) for components. The appropriate file will then be picked up at runtime. Specifying what component to render by using different file extension is a powerful concept: typically a developer would use this approach for components that will serve the same purpose but need to look differently on each platform. Both files then have the same interface which abstracts away the inner differences. Such approach can be used in a number of components such as buttons, pickers or even a non-component code.

Appendix A

Bibliography

[rn:16a] *Introducing hot reloading*, 2016.

[rn:16b] *Javascript environment*, 2016.

[rn:16c] *Native modules*, 2016.

[xam16] *Sharing code options in xamarin*, 2016.

Katedra: matematiky Akademický rok: 2008/2009

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Pro:	Tomáš Hejda
Obor:	Matematické inženýrství
Zaměření:	Matematické modelování
Název práce:	Spřátelené morfismy na sturmovských slovech / Amicable Morphisms on Sturmian Words

Osnova:

1. Seznamte se se základními pojmy a větami z teorie symbolických dynamických systémů.
2. Udělejte rešerši poznatků o sturmovských slovech: přehled ekvivalentních definic sturmovských slov, popis morfismů zachovávajících sturmovská slova, popis standardních párů slov.
3. Zkoumejte vlastnosti párů spřátelených sturmovských morfismů, pokuste se popsat jejich generování a počty v závislosti na tvaru jejich matice.

Doporučená literatura:

1. M. Lothair, Algebraic Combinatorics on Words, Encyclopedia of Math. and its Applic., Cambridge University Press, 1990
2. J. Berstel, Sturmian and episturmian words (a survey of some recent result results), in: S. Bozapalidis, G. Rahonis (eds), Conference on Algebraic Informatics, Thessaloniki, Lecture Notes Comput. Sci. 4728 (2007), 23-47.
3. P. Ambrož, Z. Masáková, E. Pelantová, Morphisms fixing a 3iet words, preprint DI (2008)

Vedoucí bakalářské práce:	Prof. Ing. Edita Pelantová, CSc.
Adresa pracoviště:	Fakulta Jaderná a fyzikálně inženýrská Trojanova 13 / 106 Praha 2

Konzultant:

Datum zadání bakalářské práce:	15.10.2008
Termín odevzdání bakalářské práce:	**7.7.2009**

V Praze dne 17.3.2009

....

Vedoucí katedry Děkan