Ellert Smári Kristbergsson

Developing multiplatform apps

Using hybrid solutions

Helsinki Metropolia University of Applied Sciences

Bachelor of Engineering

Information Technology

Thesis

2 June 2014

Helsinki Metropolia University of Applied Sciences

Author Title Number of Pages Date	Ellert Smári Kristbergsson Developing multiplatform apps 31 pages + 3 appendices 2 June 2014
Degree	Bachelor of Engineering
Degree Programme	Information Technology
Specialisation option	Software Development
Instructor	Ilkka Kylmäniemi, Lecturer

The aim of this thesis is to describe the process of developing multiplatform apps for mobile phones by discussing and comparing the different methods of developing apps for all platforms. The purpose is to ease decision making of smaller companies that are starting to develop their own product for smartphones.

The paper analyses tools based on the Cordova framework and the Qt framework. The difference between how the frameworks deliver the code to the machine, Cordova compiles JavaScript into native code for each platform but Qt compiles C++ code to binary code with different toolchains for each architecture.

Two projects, in paticular, are discussed, where an app was made with a hybrid solution. In one of the projects, an app was made for a company named Locatify with location-based interface using a framework called PhoneGap based on the Cordova framework and a project management methodology called Scrum. The other project is an app made for a company called Barnaefni for recording videos and sending them to a server.

The main results are that most of the different tools that are available for hybrid app development are based on a few tools. This makes it easy for companies with talent coming from web development to choose Cordova-based frameworks but for companies with talent from low-level programming to choose a framework based on the Qt technology.

Keywords	app, mobile, software, development, JavaScript, PhoneGap

Contents

1 Introduction

This thesis aims to compare different methods of developing hybrid apps for smartphones and describe the benefits of each solution to provide an answer to what tools and framework would be best to use when starting a new project. A brief description of the frameworks can be easy to find on the internet and since they all aim to solve exactly the same problem, that is to develop apps for all the major platforms in an easy way, it can be hard to realize the difference between them. This can be a problem for small companies when they are starting a new project because the further the project is in development, the harder it is to turn back and change the tools that they are working with.

The scope of this paper is limited to how start-up companies and developers can choose the best tools to develop an app that should work on more than one mobile platform. Two start-up companies will be discussed in particular, how they chose a framework to work with and why they chose it. One of the companies is called Locatify. They planned to develop an app based on their CMS system that existed as web applications, a treasure hunt game and a tourist guide in the same system. The other company is called Barnaefni. They wanted to build an app that could be used by children to record television material on their parents phone and send it to a server where it could be viewed by the television station.

2 Platform-independency and development

2.1 The bridge between competing standards

Standards are the basic pillars of communication. Communication consists, by definition, of five elements, the encoder (also known as the sender or the speaker), the decoder (also known as the receiver or the listener), the message, the standard (also known as the medium) and the feedback. In everyday life we encode our thoughts into words and other people listen to the words decode them into their own thoughts and give some feedback [1]. To be able to decode a message it is necessary to know the standards that were used to encode it. It works also the other way around, that is, to be able to send (or encode) a message it is necessary to use standards that are known by the receiver in order for him or her to be able to decode it and hence understand it. As an example of communication Hanna (the **encoder**) might say to Ari (the **decoder**)

“villtu rétta mér hamarinn?” (the **message**). Here Hanna has chosen the Icelandic language as a **standard** and if the decoder (Ari) does not know the standard used to encode the message, the communication will fail. However if he does know the standard, he will hand Hanna the hammer or inform her that he will (or can) not do so (since the meaning of the sentence is “could you hand me the hammer?”).

For a sender of a message or information it is important to choose the right standards. It is important to estimate how widely the standard is used within the target audience versus how hard it is to follow the standard. The more standards are fulfilled, the bigger audience group can be reached. These standards can be languages, systems of measurement, programming languages, protocols, even hardware or software. Software developers write code that is a representation of their thoughts understood by a machine that decodes it and gives feedback through the screen, the speakers or through any other output.

Software development has changed dramatically through the decades. From the time when software developers would gather in a huge room with a machine that would fill up almost the entire space to program it in assembly or even binary. Until today when software developers sit wherever they please, working together through the internet, making software for devices that can fit into people’s pockets, programming in high level, object-oriented languages.

The competing standards have always been a problem for software developers. There have always been various competing companies with their own standards and it can be confusing and frustrating to go from one standard to the other. In the 1960s it would be IBM versus UNIVAC. In the 1970s and 1980s companies such as Intel, Microsoft and Apple would come in with their own standards.

In the 1990s and early 2000s the main competing standards for software developers would be only the three, Microsoft Windows, Linux and Mac OS (Apple) which made it relatively simple for developers. Today we have again a large variety of different standards. Today’s competing standards are not only about the different operating systems and architectures but also about screen sizes, input methods and even more issues that developers, who want to make software for everyone, need to take into account.

2.2 Cross-platform solutions

To solve the problems that different standards create, software developers abstract their data in order to run their software across different platforms. The first known cross-platform program was written in December 1960. It was a program written in the COBOL programming language, compiled for two different computer architectures, the UNIVAC II and the RCA 501 [2].

Later the American Standards Association (ASA) made a standard for programming languages. At that time COBOL, FORTRAN and other programming languages would be standardized so that hardware producers could make hardware that would run software that had been programmed in languages following the standards. [3].

Even though programs could be run on many computers, each computer had their own operating system (OS) until in 1973 when an operating system called Unix was entirely programmed in the C programming language. This gave Unix a leading role in Operating systems at the time since it could be run on almost any machine. [4]

At the beginning of the 1990s operating systems with Graphical User Interfaces (GUI) became popular. Windows and Mac OS were the most popular operating systems at the time. The two systems used very different programming interfaces and therefore software would often be written for each platform separately. At the same time the internet was becoming larger and more widely used. That led to another conflict between standards often referred to as the “Browser Wars”. That war is still going on and refers to the competition between different web browsers. The difference between this conflict and earlier conflicts is that this time the standards are not defined by the hardware vendors. They are defined by the software that is built with the HyperText Markup Language (HTML), JavaScript, Flash and many other languages interpreted by web browsers. [5]

Since the standards for internet browsing are defined by the software rather than hardware, developing for different browsers has never been as big a problem in software development as developing for different devices. The difference between software on the internet and on a computer’s hard drive is constantly decreasing as bandwidth and available information on the internet grows. People play games through the internet,

store their files there, and even use word processor and spread sheet applications online.

Today it is even possible to use JavaScript and HTML to access some of the hardware connected to the end user's device. That is why web-based programming is getting more and more important. It is ideal to use when it comes to developing applications that should work on many different platforms such as desktop computers, laptops, tablets and smartphones, especially if the software is also supposed to work on many operating systems such as Android, iOS, BlackBerry or Windows.

In mobile app development, web-based solutions are very reusable between platforms and the same code base can even be used as a webpage with little or even without any changes. It is, however, often desirable to use some specifications that smartphones have and that are not available through the web standards only, such as accelerometer, compass or storage space. In that case Hybrid apps are a good solution.

A hybrid mobile app is software that is built like a website and that uses some additional framework that packs the JavaScript code into native code for each platform, enabling developers to access each phone's native Application Programming Interface (API). Without it, developers would need to write in a specific language for each platform. For example they would need to write in Java for Android, in objective C for iPhone, in C# for the Windows Phone and in C++ for Symbian.

Reusability of code is not the only reason for developing Hybrid solutions. The same origin policy that is a common problem in web development does not apply to hybrid apps. Therefore they can easily use data that is not stored on the phone from central servers. This allows developers to update their software automatically, without asking the user to download updates [6]. Some developers argue that writing everything natively increases performance. However hybrid apps have often proved to be faster due to optimisation when the JavaScript code is translated into native code which average programmers often oversee when they are writing the code natively for the phones.

3 Hybrid solution

3.1 Architecture

PhoneGap is a framework for JavaScript that enables developers to use sensors and other hardware or software utilities that are commonly available in smartphones [7]. It is the leading open-source framework for creating hybrid applications for mobile phones.

PhoneGap is often confused with Cordova but PhoneGap is actually just an addition to the Cordova framework and there is very little difference between the two. It was re-named when Adobe bought Nitobi, the creator of PhoneGap. Adobe then donated the whole PhoneGap codebase to the Apache Software Foundation. It became open source but Adobe still wanted to continue working on the project, connecting it to its closed-source products. PhoneGap is still free and open source but might offer connections to some closed-source solutions in the future. Cordova just stays as an open source project and is governed by the Apache foundation. It is a hybrid mobile framework used by PhoneGap. Cordova is named after the street where the Nitobi office used to be located. [8]

Cordova is also used by other frameworks, the most popular once being Ionic, Sencha Touch and Meteor. Ionic is a framework that tries to imitate the looks and feel of an iPhone native app. Sencha Touch is a HTML and JavaScript framework that uses mostly the HTML5 API but in the Native section of teir API documentation most of the classes use PhoneGap to access sensors and other device-specific utilities [9]. Meteor is a framework designed to make the interaction between client side and server side easy and fast in development. It is mostly focused on web development but uses Cordova to enable its users to easily change their web page into a native mobile app for any platform.

Another noticeable hybrid solution is the Qt framework. Qt, pronounced like the word “cute”, has a somewhat more interesting history than other frameworks since it was first developed in 1994, long before smartphones even existed. At that time it was focused on bridging the gap between different operating systems, namely Linux, Windows and Mac OS. As mentioned in section 2.1 these were the main three competing standards at that time. Qt was the greatest competitor of the Java programming language and its

multi-platform graphical user interface library called Swing. Since Qt is a C++ framework, all code is compiled into native binary for the machine and therefore works much faster than similar software on Java. On the other hand Java has much bigger community and libraries and it is easy to make mistakes that can lead to memory leaks in C++. [10]

The inventors of Qt also developed the Qt framework for embedded systems called Qt/Embedded and later renamed Qtopia. In 2007 they produced a mobile phone with a graphical user interface, running on Linux, called Greenphone. This made Nokia (the biggest cell phone manufacturing company in the world at that time) interested in the company and the Qt framework. In 2008 Nokia bought Qt and developed it further for their real time operating system called Symbian. [11]

Just at the time when Nokia was starting to develop Symbian with the new Qt library, Apple released iPhone. This was a bad timing for Nokia since they had just started to work on the graphical environment with Qt. When it was released it received very poor critics stating that it took too much time to learn how to use it while people could use iPhone immediately without even taking a look at the manual. The most famous example of this is when an executive of Nokia had brought home an iPhone to check out the competition and by the end of the day his four year old daughter had asked him “can I take that magic telephone and put it under my pillow tonight?” [12]. That made him realize that Nokia had a problem.

Nokia realised that they needed to change their course and focus rather on developing a Linux-based operating system called Maemo, which was merged with Intel’s Moblin platform in 2010 and then called MeeGo. MeeGo and Maemo still used Qt for the GUI. The first MeeGo phone came out in 2011 but then it was too late because Steven Elop from Microsoft had been hired as the CEO of the company and was going to collaborate with Microsoft on a mobile phone with a Windows operating system. Eventually the whole mobile phone and tablet section was bought by Microsoft and in five years Nokia had gone from being the biggest mobile phone manufacturing company in the world to not manufacturing mobile phones at all. Need to put resource!!

In 2012 Nokia sold Qt to a company called Digia. Since then Qt has been marketed as a cross-platform development tool and is rapidly growing as such. Since Qt is based on C++ all it needs is a toolchain for each platform. Therefore software developed with Qt

can run on all the major platforms (Android, iPhone and Windows Phone). Smaller platforms are also contributing to the open source part of Qt and making toolchains for their devices such as BlackBerry, Intel's Tizen and Sailfish. Today Qt offers a higher level language called QML that resembles JavaScript and CSS, so developers do not need to worry about memory leaks or optimisation. [13].

Comparing Cordova and Qt today is probably similar to comparing Swing and Qt 10 years ago. Cordova has a bigger community and more tools around it but the Qt performance is of course much better since it compiles into native binary code. Interestingly the Apahce foundation has started a project called Cordova/Qt for porting Cordova to Qt so that it will compile on all the same platforms as Qt does [14]. This means that the JavaScript, HTML and CSS from the Cordova project would compile to C++ code which would be compiled to native binary code on each device similar to the QML code. That should even be faster than apps developed with Cordova for Android since then the code is compiled to Java which is compiled to a binary for a virtual machine that needs to be interpreted to native binary code in runtime. It should perform up to the speed of apps developed with Android NDK. The problem with the Android NDK has been that it does not provide any higher level development tools like Qt does. [7][13]

Other popular multi-platform frameworks are Rhodes and Appcelerator Titanium. Both of them are free and open source but support fewer platforms, have less active community around them and poorer documentation. Also they are not based on standard web technologies like HTML5 and CSS3. In the Locatify project a hybrid solution was preferred because the app should communicate with a server. Hence PhoneGap was the obvious choice for this project. [6][7][9]

3.2 Interfaces

3.2.1 Version control

Locatify had been using a version control system called Subversion but recently started to use a system called Git. Since this was a new project, Git was chosen for version controlling and source code management. Since neither of the developers had used Git before, two hours were used for learning how to use Git and setting it up on the computers.

Git is a tool created in 2005 by the Linux community to maintain the Linux kernel, the biggest open source-project in the world. It suits big projects where many developers are working on the same code [15]. Since this project has only two developers, only a small subset of the Git tools where used in the project. The tools that were used included the following:

- **Git pull** – To copy all the files from the server to the computer where changes are going to be made
- **Git status** – To see if files that have been changed have been uploaded to the server
- **Git add** – To pick the files that are going to be sent to the server
- **Git commit -m** – To make sure that no files are overwritten and to add a comment that states what changes have been made
- **Git push** – To upload the files to the server.

3.2.2 Camera and microphone

The camera of the phone can be accessed in three ways. Cordova provides two plugins called Camera and Media Capture. They will both start the native camera application from the phone and the developer will have no control over the user interface of the camera. The main difference between the two is a JavaScript object, called options, which can be passed as a parameter to the function that starts the camera [16]. The options object has only one property called limit in the Media Capture plugin. This limit property defines how many pictures the user can take before the camera app shuts down and the screen goes back to the original app.

The options object of the Camera plugin has 12 optional parameters:

- **quality** – changes the quality versus size of the image
- **destinationType** – gets the image either from the camera or from the file system
- **allowEdit** – defines whether the user should be able to edit the picture
- **encodingType** – can be either jpeg or png

- **targetWidth** – defines how wide the image should be
- **targetHeight** – defines how high it should be
- **mediaType** – defines whether it is a picture or a video or if it can be both
- **correctOrienation** – defines whether the image should be rotated automatically to fit the screen orientation or not
- **saveToPhotoAlbum** – defines whether the picture should be saved to the device's photo album or not
- **cameraDirection** – defines whether the back or front camera will be used
- **popover** – only used for iPads to define where the camera should pop up.

The third option is to use a HTML5 standard called getUserMedia. It provides access to the device camera whether it is a mobile phone or a computer and most native mobile browsers support it. Since the native camera app has been highly optimised for every phone, getUserMedia has worse performance. Also there is a question that pops up to the user the first time he uses the app asking if he wants to allow the app to use his camera. The getUserMedia is still not so widely supported on mobile platforms. The newest Android Browser (version 37) supports it but not the browser before that (version 4.4.4 and earlier) and Safari does not support it at all [17].

The Media Capture plugin was used for the Locatify project since it has a good ready-made user interface and it provides functionality for images, sound and video unlike the Camera plugin that only provides functionality for images and video. The getUserMedi API was used for the Barnaefni project since it was supposed to work on the webpage as well as on the phone. Since it is possible to use the same code in both places, it was chosen as the best solution for that project.

3.2.3 Global Positioning System (GPS)

The Cordova Geolocation API uses the HTML5 geolocation API if it is supported by the browser. If the native browser does not support HTML5 geolocation, it will use the native API for retrieving the data. If GPS does not work on the device, it will use the wireless network to determine the geographical position. The HTML5 geolocation API is very widely supported. Therefore it is safe to assume that it will be used in most cases [18].

3.2.4 Data storage

There are several ways to store data from a mobile phone and it is important to think carefully how to store the data because it has a great effect on the performance. Many applications use Restful services. It is important to remember that mobile phones often use 3G connections that are much slower than the wireless connections that larger devices normally use. Even though 4G is becoming more common, data transmission is still much more expensive. Therefore it is good practice to store the data locally rather than querying the data over and over again. It will improve the performance, save battery power and lower the possible cost of data transmission [16].

Cordova has an API called File that reads or writes data to a file on the phone's native operating system. The file system of each phone is very different especially for Windows phones and therefore there are many different options to choose from for each device. It is still possible to write general code that works for all devices but it requires relatively many lines of code compared to the other APIs. The Cordova API is mostly based on the HTML5 requestFileSystem API and uses the same syntax.

There are three other HTML5 storage options. They are called LocalStorage, indexedDB and WebSQL. LocalStorage is a simple, synchronous key-value storage system and IndexedDB is more sophisticated key-value storage. It is asynchronous and supports transactions [19]. WebSQL is based on SQLite and provides a local SQL database. It is deprecated and the W3C working group has stopped maintaining it [20] but IOS does not support indexedDB, and therefore webSQL is still being recommended to developers by Cordova [21].

In the Locatify Project webSQL was chosen as the best option since it had the SQL interface that everyone in the team was familiar with. All queries where made instantly and the performance was still good since all the data was stored locally. On the main screen there was an option called Save that would synchronise all the data that had changed in the database since last time, to a MySQL database stored on a server.

3.3 Eclipse and the Service Development Kit

3.3.1 Eclipse

PhoneGap does not come with an Integrated Development Environment. Before version 3 the developer used to need for example Eclipse for Android, Xcode for iOS or Visual Studio for Windows. Now it is possible to use any web development environment. The Locatify project was developed before version 3 was released and therefore it was necessary to use a specific IDE for each platform.

The IDEs would be used to build the code and make an executable for each device. There would need to be a few lines of code in each language to import the Cordova library that converts the web code (HTML, CSS and JavaScript) into native. So the only preparation needed to be able to port the app to a new device would be to set up the development environment for the platform, write a few lines of code to import the Cordova library, which links to the web code, and build it.

Eclipse is a very popular IDE especially for Java developers although it can be used for any programming language. It has been open source since 2001 and is controlled by the Eclipse Foundation [22]. Eclipse has been the only IDE that supports Android development for a long time.

Locatify had released some apps for iPhone and iPad and worked with Apple products. However since Android offers all necessary tools as free and open-source, it was chosen as the platform to develop and test the product on and then it would be relatively easy to build it also for other platforms. Therefore Eclipse was the IDE that was used during the development of the product. It was, however, only used when necessary and most of the code was written in a text editor called Sublime Text.

3.3.2 Android Service Development Kit

Android SDK is a set of tools used for Android development. It includes libraries, Android Emulator, debugger and other useful software for developing apps for Android [23]. In the project, the Android Emulator was the most used feature of the SDK although the Android libraries were also used. However the Cordova framework abstracts them and therefore it was not as visible for the developer. All the debugging was done through chrome and the LogCat feature of the Eclipse IDE.

3.3.3 Eclipse alternative

Eclipse was, for a long time, the only IDE that Android developers could use. Until April 2011, when an IDE called InetlliJ started to support the Android platform [24]. Today another IDE called Android Studio is available but it has only been released for testing purposes and it is based on IntelliJ. IntelliJ was a closed-source IDE until IntelliJ released an open source version of the IDE called intelliJ Community Edition in May 2011. They did not have a user interface designer, a graphical tool that Eclipse has, until with IntelliJ version 12, released in December 2012 [25].

With the release of PhoneGap 3 it was no longer necessary to install Eclipse or any other IDE to deploy the app to an Android device. Neither should Xcode be needed for deploying it to iPhone or iPod. Adobe even has a cloud service called Phonegap Build where developers can upload the code online and build it in the cloud.

3.4 The Workflow

The Locatify project was carried out for the owner of Locatify by two developers, a student from the University of Reykjavik and the author of this paper. One of the first decisions that needed to be made was how to organize the workflow of the project. The owner of the company had been working with an agile method called Scrum. Both of the developers were interested in the method since it is very widely used in working life, especially in the branch of software development. Hence it was decided to use Scrum for the project.

3.4.1 Scrum

Scrum has proved to be one of the most effective ways of managing workflow when it comes to software development. It can be very easy to understand Scrum but mastering it can be extremely difficult since there is an enormous amount of material about it and implementing it in the right way has often been unsuccessful. To remedy this, one of the founders of Scrum, Ken Schwaber, established a website called schrum.org that aims to explain the best practice. He has also written a guide with his cofounder, Jeff Sutherland that includes the definition of Scrum. Scrum consists of roles, events, artefacts and a few rules. [26]

In Scrum there are three roles: product owner, Scrum master and the development team. The product owner is responsible for the Backlog (the document that defines the project). The owner of Locatify took the role of the product owner. The Scrum master is responsible for the Scrum process and needs to be very familiar with the system. He manages the daily meetings called Daily Scrum. It was decided that the developers would share this role and take turns in being Scrum masters, so that they would be able to get the most experience out of the project. The third role is the team members who are responsible for developing the project and finishing the tasks that are defined in the Sprint. In the Locatify project this was both of the developers.

The events are called Sprint Planning, Daily Scrum, Sprint Review and Sprint Retrospective. When a new project is started, it is divided into periods that can take up to four weeks each. The work that is estimated to be done during one such period is called a Sprint. Before each Sprint the product owner prepares a set of tasks that should be completed during the Sprint. This event is called Sprint Planning and should take about eight hours for a month-long sprint. When the Sprint is over, the team members will show the results and everyone is invited to come to see the results (customers and other stakeholders). This is called Sprint Review. In the Locatify project all other workers of the company attend those meetings. Sprint Retrospective is the last event before the next Sprint Planning takes place. Then the developers and product owner will suggest what could be done better and what was well done in the previous Sprint. In the Locatify project this was done straight after the Sprint Review.

The Scrum Artifacts consist of the Product Backlog, Sprint Backlog and the Increment. The Product Backlog defines what is needed to be done to finish the project. Normally software is never finished since there are always new versions coming out and therefore the Product Backlog is never done. The Product Backlog is a list of everything that is needed to be done for the project. The Sprint Backlog is a subset of the Product Backlog that includes the tasks that are intended to be done in the ongoing sprint. The Increment is everything that has been completed from the Product Backlog in one Sprint. This is not necessarily the same as the Sprint Backlog since often the Sprint Backlog is not totally completed during the Sprint. [26]

3.4.2 The environment

The Locatify project was carried out in a small office in Locatify's facilities in Hafnarfjörður, Iceland. The devices used for the project where two laptop computers with the Microsoft Windows operating system as well as one mobile phone with the Android operating system. One of the office's walls with A4 paper sheets and Post-it stickers was used to manage the project workflow with Scrum. A meeting room with a whiteboard was used for meetings called sprints. Sprints are a part of the Scrum methodology and include planning and reporting the status of the project to the owner and other workers.

The owner of the company needed software that would become a part of his own content management system (CMS). The company had already made a web version of the software with a Python framework called Django. What was needed was a mobile version that should run as an app on Android, iOS and Windows Phone. The product has two use cases and each of them has a name. One is called Turf Hunt, where the interface is a treasure hunt game where an organizer makes a path for the participant of the game to follow. The other one is called Smart Guide where the user creates a tour for tourists to follow on a map.

The main challenges for the mobile version of the software where to access the sensors of the phone and to synchronise the data from the phone to the database on a server provided by the company. The web version was all rendered with Django, the Python framework and therefore the mobile version could not use any of the existing code. A method to manage the project workflow was also needed and a system for version control and source code management.

The Barnaefni project was very similar but it was challenging to find out the best way to send recorded video to the server using JavaScript. It is also important to note that every time a new PhoneGap project is started, it is very important to update PhoneGap and the service development kits that should be used during the project, because it can cause conflict between the versions if it is updated after the project has been started.

4 Working apps

Since mobile app development is a very fast growing branch of the software development field, there are countless libraries and frameworks that developers have to choose from when they start a new project. It is important to do some research on the software that will be used because if it does not work as expected at some point during the development process, it can cost a lot of time and effort to change it for something else.

Even though developers can use the same code for all platforms there is certain software that needs to be installed onto the developer's machine for each platform. For Android, Android SDK is needed and Cordova uses Node.js for the backend to call the native functions of each platform. Therefor Node.js also needs to be installed. Since Android apps are normally programmed in Java, Java JDK is also needed and a build system for Java called Ant. It is similar to GNU Make for C++ but based on XML files and, like Cordova, it is a part of the Apache Software Foundation [27].

When all this has been installed, it is important to remember to either add all the binary folders to the PATH system variable or create new system variables called JAVA_HOME, ANDROID_HOME and ANT_HOME and set them to their respective values, so that PhoneGap can find them in the system. In addition to this either an Android Phone or the Android Virtual Machine is needed to be able to see the results and test the app. When Android SDK is started for the first time, it does not have any libraries installed and the developer needs to decide for what platform the app is going to be developed. Apps developed for the newest version of Android may not be backwards compatible for older versions, and apps developed for old versions of Android might perform poorly on modern phones.

Cordova uses the WebView class of the android.webkit Java pakcage. This is an Android view that displays webpages. It then uses a function called addJavascriptInterface [28] to allow the javascript to have access to some Java object. The function takes two parameters, an object and a string. The string is the name that the JavaScript can use to access the object. After it has been injected into the JavaScript its methods will be accessible through the global scope of the JavaScript by the name passed to the function [29]. In this way JavaScript (client side) can communicate with a background thread of the WebView and all communication will work as fast as if it was a real native

app [30]. A different method is used for apps targeted at Android versions earlier than 4.2 because of a security issue.

Since both Barnaefni and Locatify were start-up projects that needed to have a quickly ready app for all platforms, Cordova was an obvious choice. Rho and Native solutions would have cost much more time and the code would have needed to be rewritten for each platform. Qt was never considered an option, but since the product of the company was a webpage with a backend database and its developers were specialized in web technologies, a web-based solution was considered a benefit.

4.1 Locatify

The app for Locatify is used as a part of a CMS system for a treasure hunt game called TurfHunt and a guided tour system for educational purposes. The app was developed using the Scrum software development methodology. It took four sprints to make the app. The sprints were all named after countries in South America. The first sprint was named Argentina. It was estimated to take 49 hours, for two developers during a time period of 7 days, to finish the stories represented in the sprint. But since neither of the developers were used to the Scrum methodology they were rather optimistic and, as shown in figure 1 below, only managed to finish 30 hours.

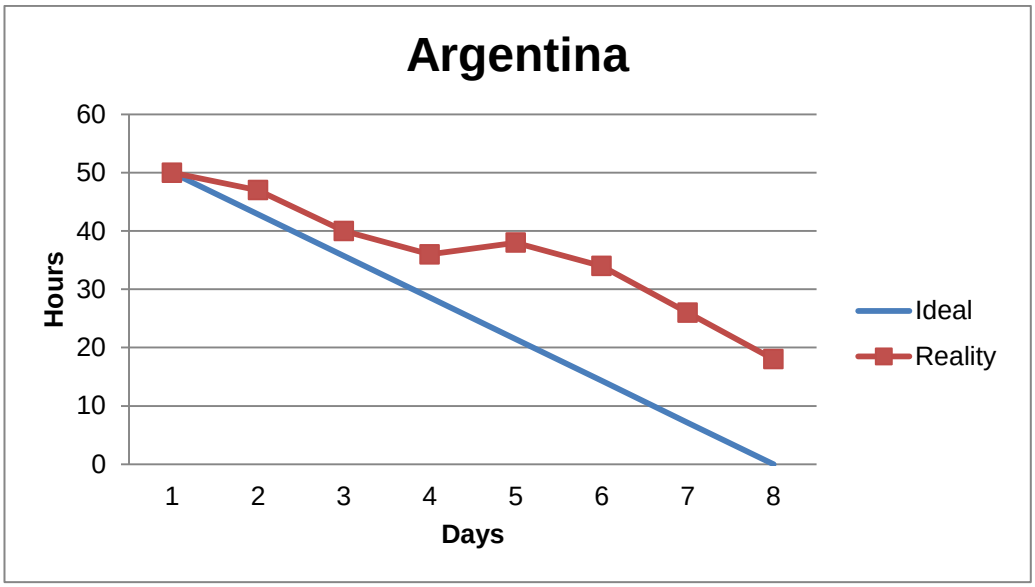


Figure 1. A burn down chart for the first sprint, called Argentina.

As figure 1 shows, the sprint was behind schedule from day 1 (Thursday 6th June). On Wednesday 12th June it became clear that a tool that had been chosen was not good for the project. Therefore, a part of the code needed to be rewritten for a new tool. That is why the line goes up between Monday the 10th and Wednesday the 12th. The main tasks of this sprint were to draw a database schema for the webSQL API that was used in the project, getting started with Git, doing some research on APIs and installing the required software. It was decided that jQueryMobile would be used for the user interface on the main screen and Google Maps would be used to create the map. jQueryMobile was used for the prototype. However, the user interface was not a priority since a graphical designer would take over the project when all the functionality was in place.

The next sprint was called Brazil. It was where the real coding actually started after all the preparation from the previous sprint. On the burn down chart there were 65 hours scheduled to be used for the sprint and the goal was to make a basic app for the tour designer and then a tour or a treasure hunting game could be made. There should be a main screen with a button. When the user taps the button, it will lead to a map view with a button at the bottom of the map. When that button is tapped, a marker will be added to the map, showing the user's current location. The user should be able to tap the flag to get a popup balloon where he could enter some text, a picture and sound or video recording about the place on the map (a point of interest for the tour guide or a point for a clue in the treasure hunt game).

The main screen displayed a jQueryMobile button, a responsive design where buttons reach over the entire width of the screen. Figure 2 shows how the buttons change in size when the user turns the phone form portrait to landscape.

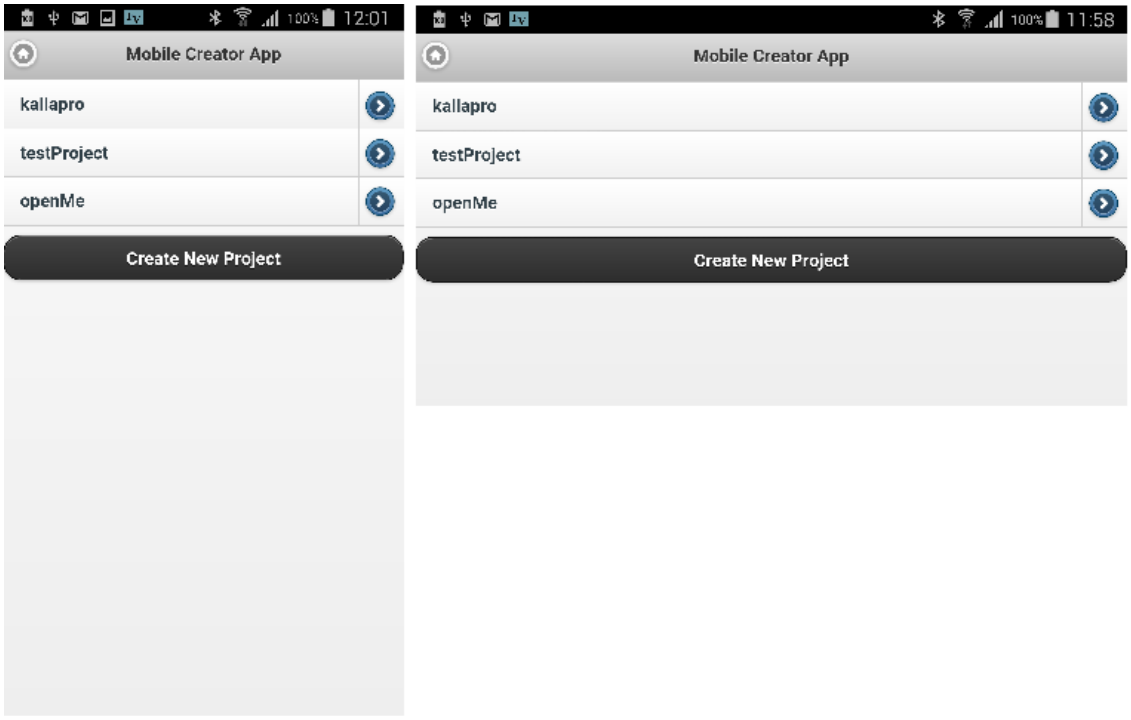


Figure 2. Main screen in portrait and landscape

As figure 2 shows, jQueryMobile changes the whole view and fits it to the user's screen. When that button was tapped, the app would change to a map view made with Google Maps API. This was done by putting all the HTML code of the main page into one HTML element called div and all the HTML code of map view into a different div. Then the display property of the CSS style for each div was put as "none" for the view that was not supposed to show and "inline" for the one that should be visible. Figure 3 shows what the map view looks like.

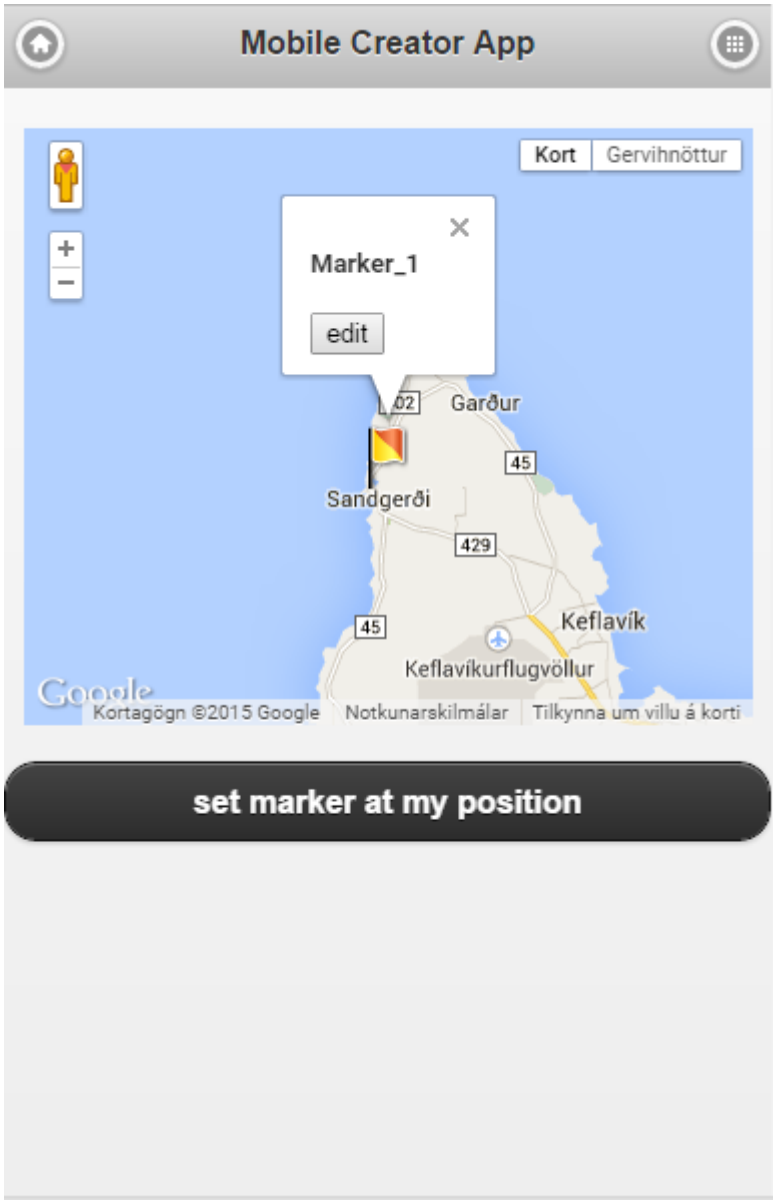


Figure 3. Map view

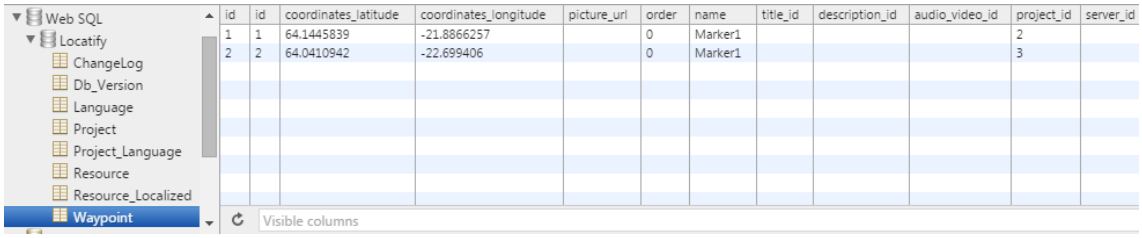
As shown in figure 3, when the user would tap a marker on the map, there would pop up a Google Maps InfoWindow. The marker is an object of the type Marker from the Google Maps API and the InfoWindow is a JavaScript type from the Google Maps API. An object of that type has a function called setContent that takes one string as a parameter. That string can include any HTML code that the developer wishes to put into the popup balloon.

The Brazil sprint started well and on schedule except that the main window was first implemented in pure HTML and was needed to be rewritten in jQueryMobile. This caused the burn down chart graph to go up (meaning that there were more hours left

than the day before) as can be seen in figure 2 in appendix 1. The jQueryMobile framework renders some new HTML elements to the page so the page looks very different from the original HTML page. This caused a few problems and made the whole sprint be behind schedule as shown in a graph in appendix 1. In the end the project had proceeded in only 25 hours out of the 65 that had been scheduled.

The third sprint was called Chile. This time the schedule was more realistic and all the participants, both the developers and the product owner, where more confident about what could actually be accomplished in one sprint. All the goals that had not been accomplished in the Brazil sprint remained as goals for this sprint and only two more goals were added. One of them was to fix the menu screen. It should show many buttons with different projects but not only one map button, so that every time a project was selected, the map view would open up with all points that had been made in that project. Also on the main screen there should be an “add new project” button that would create an empty project. The other goal that was added was about creating the webSQL database where all the points would be stored.

Adding the “take a picture” and “record sound or video” buttons to the InfoWindow was not a hard task since the operations they should perform were a part of the PhoneGap API which was easy to use and straightforward. Each call would return a URI string that represented the location of the file in the phone. This string was then stored in the database in a table called Waypoint. When the user would click the “Create New Project” button, a project name would be inserted into a table called “Project” along with an id that would be automatically generated for that project. Whenever a project was selected from the menu view, the map view would open up and all the markers that were in the Waypoint table along with their sound, video, picture and text would be rendered onto the map. Figure 4 shows the Waypoint table in the database.



id	id	coordinates_latitude	coordinates_longitude	picture_url	order	name	title_id	description_id	audio_video_id	project_id	server_id
1	1	64.1445839	-21.8866257		0	Marker1				2	
2	2	64.0410942	-22.699406		0	Marker1				3	

Figure 4. The Waypoint table

As figure 4 shows, the database chosen for the project was the webSQL where one SQL file, written in SQLite, was fetched from the server with AJAX, and a function

called process from the webSQL API was called to execute the SQL code in the file. This was not the only way to execute SQL statements but it was a good way of executing longer statements. Statements like “SELECT”, “INSERT INTO” and “UPDATE” were all written as a string in the JavaScript code and passed as an argument to the process function from the webSQL API. The code in the SQL file created the tables needed for the project. It seemed to work flawlessly at first until the app was turned off and turned on again. Then the SQL file would run again and try to create new tables with the same name, which resulted in an error. The database was not accessible anymore so there would be no projects rendered to the screen. This was fixed by adding “IF NOT EXISTS” after the “CREATE TABLE” statements to each table. It would probably have been even better just to try to select data from one of the tables and if it had not worked, to execute the code from the SQL file. In that way the AJAX function would not have needed to be called every time.

Since everything that the user did depended on the database, there would be an extreme number of different SQL statements that would be needed to be executed. So it was decided to make a separate JavaScript file, called database.js, with a separate function for each different call to the database. This made the code in the main JavaScript file much more readable. The code in database.js was always rather unorganised and could have included much fewer functions. The functions could then have taken an object as a parameter and used it like function overloading is used in most other programming languages. That is a common replacement for function overloading in JavaScript. The file database.js had functions such as addProject, deleteProject, getProject, getWaypointsInProject or updateWaypoint.

When the app was started, all the projects would be fetched from the database and a button representing the app would be rendered to the main page. When a project was selected from the list of projects, it would query the database for all markers belonging to that project id (in the Waypoint table). The markers would be rendered on the map and all the data that would be in the InfoWindow that pops up when a marker is tapped, would be rendered there but it would be invisible until a marker has been tapped on the map. This was no problem with components made with jQuery or pure JavaScript but the buttons on the main screen were made with jQueryMobile and did not show up when rendered to the page. The reason was that to be able to dynamically render jQueryMobile elements it is necessary to call a refresh function. The refresh function is

not called by the user of the API but passed as a string argument to a function called listview.

The burn down chart for the Chile sprint looked much more like the samples of burn down charts in books about SCRUM, as shown in the graph in appendix 2. It looked more like an ideal burn down chart than the ones for the Brazil and Argentina sprints. The progress line goes over and under the ideal line as expected and ended 6 hours above the ideal line. So the hours completed were 68 hours in total out of 74 hours scheduled for the sprint. There was one task about cleaning up the code, so that it would be easier to use it in the next sprint. The task was not considered totally ready and therefore 6 hours were remaining.

The last sprint was called Dominique Republic. There were four main goals for that sprint. One was to make it possible to create a project with a specific language. Another goal was to add new views, one to see the list of markers on the map because sometimes when the map is zoomed out, two markers can be so near to each other that it is hard to tap the correct one. Another goal was to edit markers so that a marker could be edited on a full screen view and the InfoWindow would only display the content of the marker and have one edit button. The third goal was to make a small navigation bar on top of the page where the user could navigate to the project view, the map view or the edit marker view. The fourth goal was to be able to upload the project to a central server where it would show on the webpage in the companie's own CMS and be editable there as well.

It is important to be able to create a project in a specific language because tour guides want to be able to make one tour for English-speaking people and another one for French-speaking. To start with it was decided to have only English and Icelandic as choices and then, if that worked, it should be easy to add more languages. Each project should be able to have many languages and each language should be able to be in many projects. Therefore, the relations in the database were many to many, and a special table was created with two foreign keys: one for language id and another one for project id. Then there was the language table (including only Icelandic and English at the time). An arrow was added to the project buttons on the main page that would open up a dropdown menu under each project where the user could rename the project and add a language if needed. There was also an upload button added to upload the project to the CMS server. Two lines, inserting English and Icelandic into the language

table, were also added to the bottom of the SQL file, which runs every time the app is started. To prevent the app from showing errors it, the statement that was used was actually an “insert or update” statement even though it was never updated.

Adding the edit marker view was relatively easy since all the buttons there were static and it did not have any dynamically rendered content like both of the other views. Figure 5 shows how the looks like.

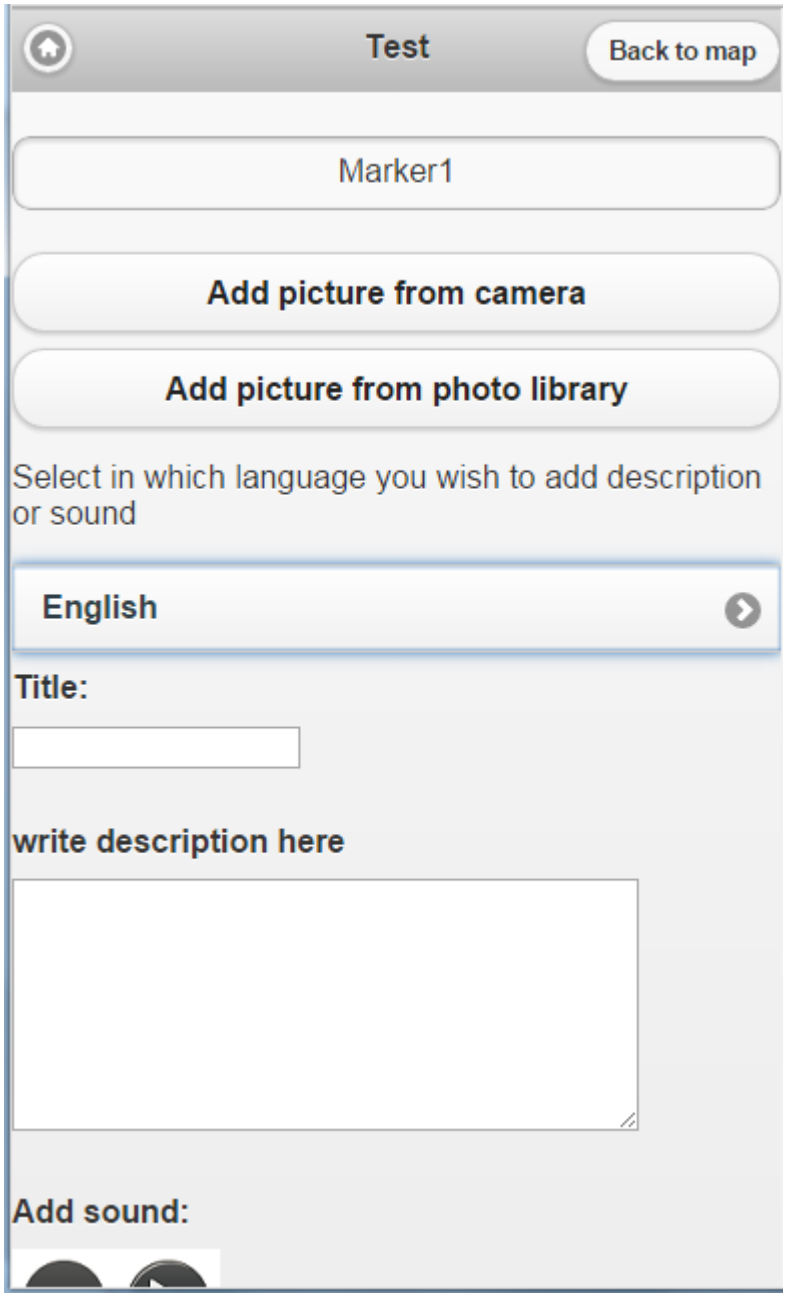


Figure 5. Edit marker view

As figure 5 shows all these options took too much space to be shown in an InfoWindow. However, after taking all the buttons out of the InfoWindow it looked relatively empty with only one edit button and the name of the marker on top, when no content had yet been put into it. So it was decided to make a default picture that would show that there was no content there yet. The marker list was dynamically rendered by selecting all markers from the current project. This was harder than the InfoWindow since the current project was not available in the scope, so it was decided that when a project would be selected, the div element that would include the viewpoints would get an attribute with the project id. This way it was available in the global scope. The result is shown in figure 6 below.

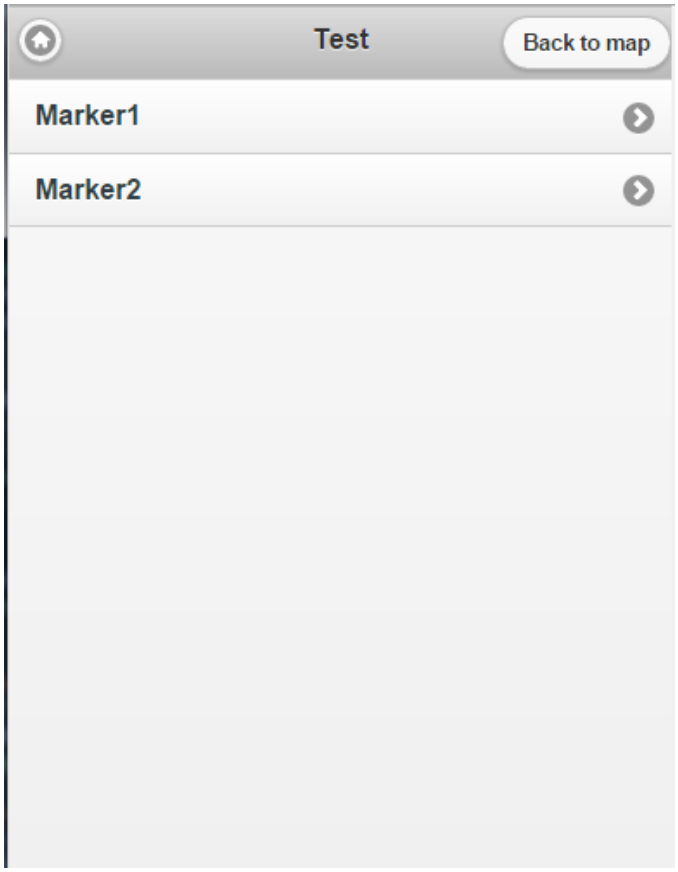


Figure 6. Marker list view.

The navigation bar was challenging from the user interface perspective. It could not always have the same buttons since it could not have a map view in the main view. It was not possible to know which map to show before a project had been chosen. Therefore, the navigation bar has only a home button on the menu view as can be seen in figure 2. It could not have an edit marker view on the map view since it was not known

which marker the user would like to edit. So it was decided that in the upper left-hand corner there would always be a home button that would take the user to the menu view. In the middle there would be the name of the project except in the main view where it would be empty. In the upper right-hand corner there would be either a menu button that would take the user to the view with a list of projects (seen in figure 3), or a button that says “back to map” (seen in figures 5 and 6). When the user changes from one view to another, all the div elements are set to hidden except the view that the user is navigating to, but that is set to show through jQuery. It basically toggles between setting the display as none and inline.

A RESTful API created with the Python framework Django was used to upload the project to a server. Programmers from the company provided the API but they had been using the Django framework to make the webpages for the CMS. To use the API all that was needed was to send an AJAX POST request to the server with an object that included a string called action and an object called data. The string can be add_project, update_project, add_point, update_point and delete_point. The object includes information about the action. For example, if the action was update_point, the data should include project_id, waypoint_id, waypoint_name, latitude, longitude, radius, index, and an object called locales that would tell in what language the marker should be.

Since the user might have created a project and added some markers to it, it is important that when the project is uploaded to the server, everything will be done in the correct order. Therefore, a table was created in the database called ChangeLog. It had columns called field_object that would be the data object to be passed to the POST request, action type to be passed as the action, timestamp, uploaded and id. When the upload button was pressed, all the actions in the change log would be uploaded to the server. When the server responded with the number 1, it would mean that the transfer was successful. Then the uploaded column would be updated to 1, meaning that it had been uploaded to the server and therefore will not be uploaded next time the upload button is pressed.

There were some problems with the server side at first but after a few e-mails it looked like the only problem was the same-origin principle. It took a while to find out what it was, but in the end it was only that the final “/” of the http address that the post request was sending to was missing. When the final sprint was near to an end, there was only the testing left, and for some reason the recording of a video had stopped working dur-

ing the process. The problem was that when the developer who had added the video feature, had pushed it to Git, the other developer had forgotten to execute the pull command before making some other changes to the main.js file. Then he had pushed some new version with some other changes, and nobody had noticed that the function for recording videos was not there anymore.

This resulted in the sprint being behind schedule by 20 hours. As shown in the graph in appendix 3, the result of the final sprint was that 46 hours of work had been done, out of the 66 that had initially been scheduled for the sprint. Since one of the developers was going back to school and the other one was going to work after the final sprint, they needed to meet up a few times at the University of Reykjavik to finish the few bugs that still needed to be fixed.

4.2 Barnaefni

The Barnaefni app only used the getUserMedia API from PhoneGap but then it sent the recorded video to the server through AJAX. Each frame of the video was changed into a JavaScript object called a blob containing a jpeg formatted picture representing the frame and sent to the server where a very simple PHP script received it and saved it as a mp4 file. The app worked although the quality of the video was poor because the files were required to be small, so that the video could be sent through 3G without high costs for data transmission. This could probably be fixed by using the WebRTC standard to send the video through socket.io.

The Barnaefni app has never been published and will never serve the purpose that it was meant to do since the company is now focusing on tourism and has abandoned its ideas about the television station. There are, however, some ideas about using the app as a tourist app where tourists could record a video where would tell their stories about visiting Iceland. There have also been some ideas about making an app where people could send their resume to a company in the form of a video and the company would help other companies and organisations in the creative branch to find employees.

5 Choosing the best tools

Strart-up companies normally have limited sources of talent and time and often they are not able to do the necessary research on different libraries, frameworks or other tools that they can use to accomplish their goal. Therefore they often start a project using tools that are wrong for them. This can result in unnecessary work for the developers, poor control over the project or, in the worst-case scenario they might not be able to proceed with the project and need to start over from the beginning. This thesis has discussed the options that small companies with limited resources have when choosing a platform focusing on open source software that can be optimised for the needs of each project. The thesis also discussed the possibilities of free but closed-source software that often uses the open-source solutions with some addition and often an option to pay for some services related to the software.

The workflow and talent needed for the project depend greatly on the tools chosen for developing the product itself. At first it looks as if there were a wide variety of products offering multiplatform solutions. Ionic, Sencha Touch, Meteor and many other products look as if they would be offering their own solution but they all rely on Cordova to compile the JavaScript into native code for each platform. In particular Cordova and another framework called Qt that uses different methods than Cordova were discussed and compared. The results are that Qt is a good choice for projects that want to be able to optimise their code and have full control over each platform since it is based on C++ code that is compiled for each platform. It gives developers full control with low level programming.

Cordova, on the other hand has, as mentioned in chapter 3.1, many more tools for optimising the most common operations. It has as well an easy to make, but good looking user interface. It can be practical to use these tools since they make it easy to imitate industry standards for user interfaces, such as Ionic that imitates the looks of a native iOS app and others that offer a wide variety of themes. On the other hand it can also bring unexpected results to the code since UI-based frameworks often render some extra HTML elements to the page.

In smaller companies it is also important to estimate the best way to organise the workflow. Comparing the two projects discussed in this thesis it can be concluded that project management systems like Scrum suit bigger teams, such as the one in the Locatify

project. It can, however, be hard to implement and it can be challenging for the team to estimate what is realistic for them to accomplish in a certain time-frame. When everyone involved is realistic about what they and other members of the team can accomplish, it is very useful and helps in estimating what can be done. In smaller groups such as the Barnaefni project it would have only slowed the project down.

The comparison of the different tools and methodologies discussed in this thesis can help small start-up companies to make important decisions. Many articles and blogs have been written trying to compare different tools but often they fail to realize that in the core they are often based on the same technology. Reading this thesis can help start-ups that are planning to make a smartphone app on deciding how to start the project and ease their decision making based on the talent existing in the company and the requirements for the product.

6 Reusability and further development

The Locatify app is a part of a CMS system called “Creator CMS” where people can create treasure hunting games or guided tours and allow other people to play them on their devices. Even after the developers who developed the app for Android have left the project, the app is now also available for iOS. In addition, compiling the app for iOS should not have been hard because none of the code needed to be rewritten. Most of the code functions on a website but the camera, microphone and GPS are all sensors that need native code for accessibility. The PhoneGap framework abstracts this away from the native code and no changes are needed to compile the code into a native app for each platform [31].

The app could be further developed to use the compass to point at certain buildings to display information about the buildings or clues in the treasure hunt game. This can be done by using simple calculations to see if a point is on a given line. The line is defined by the coordinates of the phone and the corner given by the compass and the point that needs to be checked is the coordinates of the building. It might be hard to be so exact as to hit the precise spot where the building is, but it is also possible to see if the line lies between two points that could be for example 0.2 longitudes and latitudes away from the actual point in both ways.

The Barnaefni app offers grater possibilities for further development. It could be used for recording short videos of people searching for jobs, for example actors, performers or people who need to be able to present themselves in front of a crowd or a camera. It could also be used in the tourist branch for people to send in reviews about places that they have visited. Some ideas related to the tourist branch came up. One idea was to make an app where people could go to places like restaurants, cafés and museums and when they have visited a certain number of places they could get discounts or some other prizes. It could be either location-based or every place would have a QR code that could be scanned to earn points.

There have also been some ideas about making new apps based on these technologies. For example an app that could count the hours a person is working by sensing when a person has spent more than an hour at his or her working place, counting how long the phone is there and adding it all together until the end of the month or the week. All the user would need to do is to mark the working place on the map and enter the days when the salary is paid. This would save people time in writing down their working hours as so many people still do today.

Another idea has been for people who go fishing or hunting. If they have a good day, they could take a picture of the weather conditions and the location and the date would appear automatically and then there could be an option to enter some notes about what bait was used and so on. In that way they could come back next year and see if it was just luck.

References

1 Kaul A. Effective Business Communication. Prentice-Hall of India Private Limited, M-97, Connaught Circus, New Delhi-110001, Asoke K. Ghosh; 2006. p.2.

2 Ceruzzi PE, A History of Modern Computing. Salisbury, United Kingdom, Techset Composition Ltd.; 2003. p.92.

3 Adams JC, Brainerd WS, Hendrickson RA, Maine RE, Martin JT, Smith BT. The Fortran 2003 Handbook: The Complete Syntax, Features and Procedures; 2003. p.2.

4 Milo. History of operating systems [online]. Tustin, California, United States of America, 92781, Milo; 3 October 2010. http://www.osdata.com/kind/history.htm. Accessed 29 March 2015.

5 Golding P, Shuen A. Connected Services. The Atrium, Southern Gate, Chichester, West Sussex, P019 8SQ, United Kingdom, John Wiley & Sons Ltd; 2011.

6 Rhomobile Inc. “hybrid applications” and Rhodes [online]. 3031 Tisch Way #1002 San Jose, California 95128, United States of America, Rhomobile Inc.; 23 September 2011. http://www.rhomobile.com/blog/hybrid-applications-and-rhodes/. Accessed 5 July 2014.

7 Adobe PhoneGap Enterprise. Easily create apps using the web technologies you know and love: HTML, CSS, and JavaScript [online]. 345 Park Avenue, San Jose, United States of America: Adobe Systems Inc; 9 May 2014. http://phonegap.com. Accessed 2 June 2014.

8 Leroux B. Phonegap, Cordova, and what’s in a name? [online]. 345 Park Avenue, San Jose, United States of America: Adobe Systems Inc; 19 March 2012. URL: http://phonegap.com/2012/03/19/phonegap-cordova-and-what%E2%80%99s-in-a-name/. Accessed 18 July 2014.

9 Sencha Inc Ext.device.Accelerometer [online]. 1700 Seaport Boulevard, Suite 120, Redwood City, California, United States of America, 94063, Sencha Inc; 21 October 2014. http://docs-origin.sencha.com/touch/2.4/2.4.1-apidocs/#!/api/Ext.device.Accelerometer. Accessed 29 March 2015.

10 Dalheimer M K. Qt vs. Java [online]. Norrings väg 2, 683 31 Hagfors, Sweden, Klarälvdalens Datakonsult AB; 14 February 2005. http://turing.iimas.unam.mx/~elena/PDI-Lic/qt-vs-java-whitepaper.pdf. Accessed 29 March 2015.

11 Fitzek FHP, Mikkonen T, Torp T. Qt for Symbian. The Atrium, Southern Gate, West Sussex, P019 8SQ, United Kingdom, John Wiley & Sons, Ltd; 2010.

12 Malkavaara L. This is how a Helsingin Sanomat journalist tried to save Nokia. Töölönlahdenkatu 2, Helsinki Finland, Sanoma Media Finland Oy; 2013.

13 The Qt Company Ltd. Community Supported Platforms. Tianfu, Software park No.801, Tainfu Avenue, Mid Section, Chengdu, China, The Qt Company Ltd; 2015.

14 Kinard M. cordova-qt [online]. Apache foundation; 30 May 2014. https://github.com/apache/cordova-qt. Accessed 30 March 2015

15 Chacon S. Pro Git. Spring Street, 6th Floor, New York, United States of America, Springer-Verlag; 2009.

16 Apache Cordova. Next Steps [online]. 1901 Munsey Drive, Forest Hill, Maryland, Unitet States of America, Apache Foundation. 28 October 2014. http://cordova.apache.org/docs/en/4.0.0/guide_next_index.md.html#Best%20Practices. Accessed 30 March 2015.

17 Deveria A. getUserMedia/Stream API [online]. Alexis Deveria; 21 February 2015. http://caniuse.com/#feat=stream. Accessed 30 March 2015.

18 Deveria A. Geolocation [online]. Alexis Deveria; 21 February 2015. http://caniuse.com/#feat=geolocation. Accessed 30 March 2015.

19 Bell J, Graff E, Mehta N, Orlow J, Popescu A, Sicking J. Indexed Database API [online]. W3C; 8 January 2015. http://www.w3.org/TR/IndexedDB/#transaction-concept. Accessed 30 March 2015.

20 Hickson I. Web SQL Database [online]. W3C; 18 November 2015. http://dev.w3.org/html5/webdatabase/. Accessed 30 March 2015.

21 Adobe PhoneGap Enterprise. Storage [online]. 345 Park Avenue, San Jose, United States of America: Adobe Systems Inc; 27 November 2014. http://docs.phonegap.com/en/edge/cordova_storage_storage.md.html#Storage. Accessed 30 March 2015.

22 Burnette E. Eclipse IDE Pocket Guide. Gravenstein Highway North 1005, Sebastopol, California, United States of America, O'Reilly Media, Inc.; 2005.

23 Android. Tools Help [online]. Google Inc; http://developer.android.com/tools/help/index.html. Accessed 30 March 2015.

24 Toporov E. IntelliJ IDEA 10: free IDE for Android development [online] Wordpress; 19 October 2010. URL: http://blog.jetbrains.com/idea/2010/10/intellij-idea-10-free-ide-for-android-development/. Accessed: 3 August 2014.

25 Cheptsov A. IntelliJ IDEA 12 is Abailable for Download [online]. WordPress; 5 December 2012. URL: http://blog.jetbrains.com/idea/2012/12/intellij-idea-12-is-available-for-download/. Accessed: 3 August 2014.

26 Schwaber K, Sutherland J. The Scrum Guide [online]. One Breadway, Cambridge, Massachusetts, United States of America, Scrum Inc; 16 December 2014. http://www.scrumguides.org/docs/scrumguide/v1/Scrum-Guide-US.pdf#zoom=100. Accessed 30 March 2015.

27 Apache Ant. Welcome [online]. 1901 Munsey Drive, Forest Hill, Maryland, Unitet States of America, Apache Foundation; 23 May 2014. http://ant.apache.org/. Accessed 30 March 2015

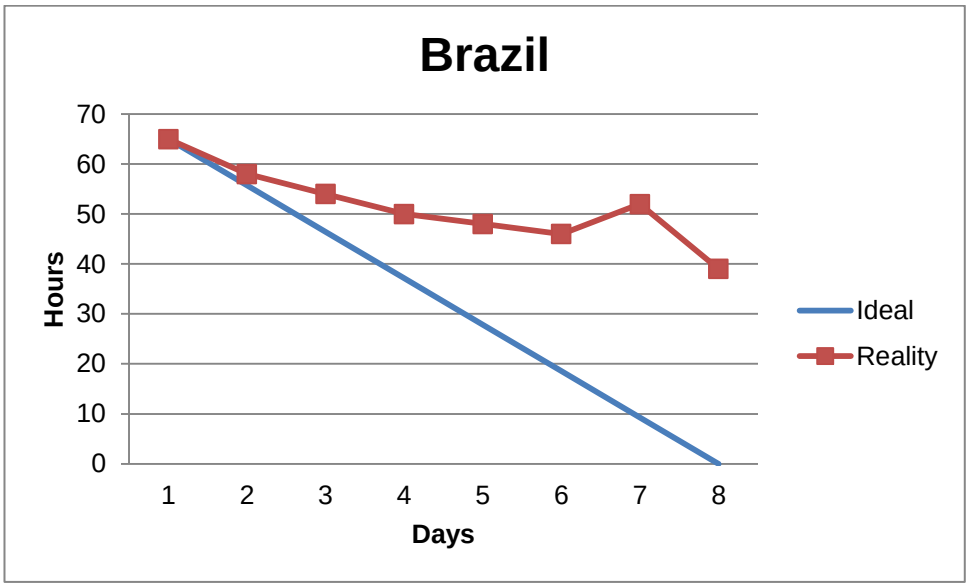
28 Grieve A. SystemWebviewEngine.java [online]. Apache Foundation; 19 February 2015. https://github.com/apache/cordova-android/blob/master/framework/src/org/apache/cordova/engine/SystemWebViewEngine.java. Accessed 30 March 2015. Line 237.

29 Grieve A. nativeapiprovider.js [online]. Apache Foundation; 18 March 2015 https://git1-us-west.apache.org/repos/asf?p=cordova-android.git;a=blob;f=cordova-js-src/android/nativeapiprovider.js;h=2e9aa67bb6a667ef9e986d3c3691078d3d0cb51f;hb=8d5cb00bec3f3cac28ed9a30cba2fedac7a0a317. Accessed 30 March 2015. line 24

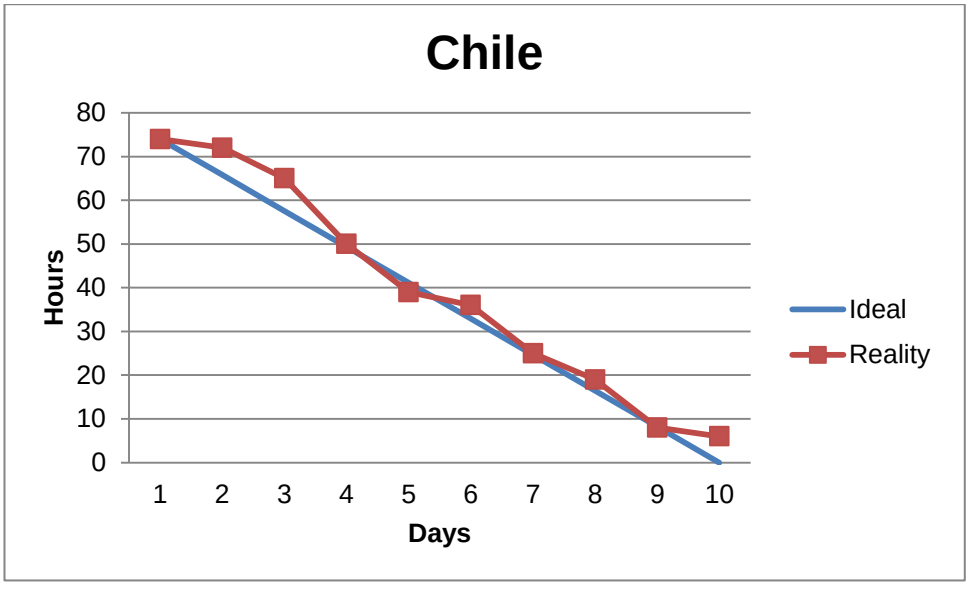
30 Android. WebView [online]. Google Inc; http://developer.android.com/reference/android/webkit/WebView.html#addJavascriptInterface(java.lang.Object, java.lang.String). Accessed 30 March 2015.

31 Björnsson LB. Creator CMS [online]. Strandgata 31, Hafnarfjörður, Iceland, Locatify; https://locatify.com/creator-cms/. Accessed 30 March 2015.

Appendix 1: Burn down chart for Brasilia sprint



Appendix 2: Burn down chart for Chile sprint



Appendix 3: Burn down chart for Dominique Republic sprint