Universidade de Aveiro
2013
Departamento de Electrónica, Telecomunicações e Informática (DETI)

Rui Mota
Bertão Lemos

Desenvolvimento de aplicações para smartphone usando tecnologias web
Smartphone applications development using Web Technologies

Universidade de Aveiro 2013

Departamento de Electrónica, Telecomunicações e Informática

Rui Mota
Bertão Lemos

Desenvolvimento de aplicações de smartphone usando tecnologias web
Smartphone applications development using Web Technologies

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia de Computadores e Telemática, realizada sob a orientação científica do Prof. Doutor Hélder Zagalo do Departamento de Electrónica, Telecomunicações e Telemática.

Dedico este trabalho aos meus pais, irmão e avó pelo apoio constante ao longo do meu percurso académico e à minha namorada Carla pelo amor e força que me deu sempre que mais precisei dela.

O júri / the jury

Presidente / President	Prof. Dr. Joaquim Manuel Henriques de Sousa Pinto Professor Auxiliar da Universidade de Aveiro
vogais / examiners committee	Prof. Dr. Fernando Joaquim Lopes Moreira Professor Associado da Universidade Portucalense
	Prof. Dr. Hélder Troca Zagalo Professor Auxiliar da Universidade de Aveiro (orientador)

Agradecimentos

Quero agradecer ao meu orientador o Professor Hélder Zagalo, pela ajuda ao longo desenvolvimento da minha dissertação especialmente no que toca a manter-me focado nos objectivos que propus a mim próprio.

Quero agradecer também à PT Inovação por me ter dado oportunidade de realizar um projecto que dá tema a esta tese, mas quero agradecer especialmente à minha orientadora na PT Inovação, Engenheira Telma Mota, pela excelente integração que fez de mim na empresa e pelo acompanhamento que me deu ao longo do desenvolvimento do meu projecto.

Agradeço a toda a minha família, mas um obrigado especial aos meus pais, ao meu irmão Tiago e à minha avó Lela, por estarem comigo neste percurso desde o início

Agradeço também a todos os meus amigos e colegas por me terem acompanhado em todo o meu percurso académico, tornando-o inesquecível. Mas deixo aqui um agradecimento especial ao Tiago pelos conselhos que me deu durante o meu último ano e ao Lucas, que por felicidade de circunstâncias, me ajudou bastante nesta recta final.

Quero dar um agradecimento muito especial à minha maravilhosa namorada Carla, pois foi uma grande fonte de motivação, apoio e sobretudo amor.

Por fim quero dar um agradecimento ao meu falecido Avô Quim, pois a forma como me ajudou a educar e os valores que me ensinou, sempre vão ser para mim algo que nunca vou esquecer.

Palavras-chave

HTML 5, Javascript, CSS, Android, iOS, jQuery, jQuery Mobile, Ext-JS, Sencha Touch, aplicações moveis híbridas, programação móvel.

Resumo

O mercado dos smartphones nos últimos anos tem vindo a crescer de forma acentuada e com isso as possibilidades para programadores de dispositivos móveis também aumentaram. Este mercado em expansão, é actualmente dominado por dois Sistemas Operativos: Android e iOS. Embora estes dois tenham uma grande quota do mercado (91% das vendas de smartphones em 2012 foram de um destes dois sistemas) é do interesse de um programador desenvolver aplicações que sejam compatíveis com o maior número de dispositivos e sistemas operativos possíveis.

Empresas como a PT Inovação têm portanto grande interesse neste mercado e pretendem ter ao seu dispor as melhores ferramentas possíveis para conseguir desenvolver aplicação capazes de atrair clientes. Como tal esta dissertação partiu de uma proposta da PT Inovação que consiste no estudo de um método de desenvolvimento de aplicações móveis, que torne mais fácil a criação de uma aplicação móvel compatível com vários dispositivos e sistemas. O método referido é o desenvolvimento de aplicações móveis baseadas em tecnologias web (HTML5, CSS3 e JavaScript). Para estudar este método foram desenvolvidas várias aplicações móveis híbridas, que se baseavam em aplicações já desenvolvidas nativamente pela PT Inovação. Após esta fase de desenvolvimento, foi feito um estudo comparativo entre as aplicações híbridas e as aplicações nativas, onde se observou quais apresentavam uma melhor performance, uma maior facilidade desenvolvimento e um design mais apelativo para os vários elementos que constituíam as aplicações. As conclusões deste estudo são o resultado principal da dissertação.

Keywords

HTML 5, Javascript, CSS, Android, iOS, jQuery, jQuery Mobile, Sencha Touch, hybrid mobile applications, mobile programming

Abstract

The smartphone market in recent years has grown sharply and with it the possibilities to mobile devices programmers have also grown. This expanding market is currently dominated by two operating systems: Android and iOS. Although these two have a big share of the market (91% of the smartphones sales in 2012 were from one of these two systems) is in the programmer interest to develop applications that are compatible with the largest number of devices and operating systems possible.

Companies like PT Inovação have great interest in this market and want to have at their disposal the best tools available to develop application that can attract customers. As such, this dissertation is a study, proposed by PT Inovação, of a method for the development of mobile applications, which makes easier to create applications compatible with several devices and systems. The aforementioned method is the development of mobile applications based on web technologies (HTML5, CSS and Javascript). To study this method several mobiles hybrid applications were developed based in applications already developed, by PT Inovação, using native technology. After the development phase, it was made a comparative study between the hybrid and the native applications, to comprehend which ones had better performance, were easier to develop and had more appealing design, for the several elements that constituted the applications. The conclusions of this study are the main result of the dissertation.

Contents

List of Figures

List of Tables

List of Acronyms

A
AJAX – Asynchronous JavaScript and XML
API – Application Programming Interface
AR – Augmented Reality

C
CAGR – Compound Annual Growth Rate
CDN – Content Delivery Network
CSS – Cascading Style Sheets

D
DOM – Document Object Model
DPI – Dots per Inch

G
GPL – General Public License
GPS – Global Positioning System

H
HDPI – High Dots per Inch
HTML – Hypertext Markup Language

J
JQM – jQuery Mobile
JS – JavaScript
JSON – JavaScript Object Notation

L
LDPI – Low Dots per Inch

M
MDPI – Medium Dots per Inch
MVC – Model View Controller

O
OEM – Original Equipment Manufacturer
OS – Operating System

P
POI – Point of Interest

R
REST – Representational State Transfer

S
SASS – Syntactically Awesome Style sheets
ST – Sencha Touch

U
UI – User Interface
URL – Uniform Resource Locator

W
W3C – World Wide Web Consortium
WHATWG – Web Hypertext Application Technology Working Group
WP8 – Windows Phone 8

X
XHDPI – Extra High Dots per Inch
XML – eXtensible Markup Language

1. Introduction

1.1 Context

Nowadays the web is very important in the life of most people, either to navigate on the web, play a game or just staying in touch with the world. Therefore the technologies inherent to the web have greatly evolved in the last 15 years (since the arrival of the version 4 of HTML [1]) and with that a lot of possibilities for developers.

Smartphones are a great tool which allows everybody to always be online. These devices have seen a huge increase in sales in the last 3 years (60% CAGR [2]), which shows clearly the trend that people want to do more than just call someone and send SMS with their mobile phones.

Similar to the web, also smartphones have changed a lot in the last 3 years, evolving from smartphones with single cores, low processing power and poor connectivity, to real pocket computers. This evolution opened the door for programmers to develop more complex applications, which make a better use of the devices that they are targeting and therefore offer the users more appealing options in terms of what they can put in theirs phones.

Despite this, developers still encounter a lot of problems when developing applications, being one of them realizing how to reach a bigger audience with the product they develop. Nowadays, the smartphone market is mainly controlled by Android and iOS (with 68,8% and 18,8% of the market share in 2012 respectively [3]), therefore is only logical that developers focus their attention in making applications for both platforms. However that can be somehow problematic to achieve, since developing for Android (Java) is considerably different from developing for iOS (Objective-C) and therefore the same application has to be developed two times, which can be very time consuming. Furthermore some developers might want to dwell in Windows Phone or Blackberry world, among others, and that will take more time and a different set of skills.

To solve this problem, one solution could consist of hiring a development team for each targeted platform developing the same application, but this will require more

resources that sometimes are not available. An alternative solution would be to resort to Web technologies like HTML5, CSS3 and JavaScript and develop applications that will be able to follow the "Develop once deploy anywhere" philosophy. This means that an application which is written using web technologies will be able to run in Android, iOS, Windows Phone, etc., with minimal code changes. These kinds of applications are known as hybrid, because they use web technologies but are packaged in a native container.

This dissertation is mainly focused in studying web frameworks that are used nowadays for the development of hybrid mobile applications, in this case jQuery Mobile (JQM) and Sencha Touch (ST). From this study, it will be possible to draw some conclusions about performance, ease of learn and which frameworks are more suitable to each type of applications or if, on the other hand, native technologies are still the best option.

This study was proposed by the company PT Inovação. The company has a great interest in understanding how viable these tools are and in which way they can be used in a development of smartphone applications. To help in the study, PT Inovação allowed access to some of their native developed applications, which then were replicated with web technologies. The full access to professionally developed applications, grants this study with a depth and detail that could not be achieved otherwise.

1.2 Purpose

The purpose of this work is to study the two most used HTML 5 mobile application frameworks (Sencha Touch and jQuery Mobile), conclude what advantages each one of them offers in different types of functionalities and if developing with web technology is in fact viable compared to developing natively. To conduct this study, a native Android application was replicated using the HTML5 frameworks. This application has some simple features, ideal for an introductory study of the frameworks.

Based on the study conducted, a more complex smartphone hybrid application was then developed, which used the framework that best fit the development process in that particular case. This development phase will serve as a proof of concept of the conclusion gathered in the study.

1.3 Structure

The dissertation will have the following structure:

- In **Chapter 1** the context, purpose and structure of this dissertation is presented, in order to give an overview and contextualization of the work developed;

- **Chapter 2** addresses all the concepts used throughout this dissertation: smartphone market, developer environment of mobile applications, evolution of web technology, overview of Sencha Touch and jQuery Mobile.

- **Chapter 3** talks about the native application that will be reproduced using Sencha Touch and jQuery Mobile. The purpose of this chapter is to display the main functionalities present in the native application, which were replicated using the HTML5 frameworks.

- In **Chapter 4** the implementation of the native application using the HTML5 framework is presented. This chapter focuses on the main functionalities developed and it includes the conclusions about the HTML5 frameworks, which were drawn during the process of developing the application.

- In **Chapter 5** some good programming practices and tricks when using HTML5 are explained. These tricks are mainly focused in improving the performance of mobile applications, so they feel native as much as possible.

- **Chapter 6** starts by explaining the motives that led to the choice of the framework used in the second application. Then the application development process is explained, followed by some conclusion about the results obtained.

- In **Chapter 7** the work done is summarized, the final conclusions are drawn and future work that can be done in order to explore more this topic is presented.

2. Mobile Web

2.1 Smartphone Market

During the last 6 years (since the appearance of the first IPhone), the smartphone market has been growing rapidly. From 139.288 thousands devices sold in 2008 [4] to 1.746.176 thousands in 2012 [5], the growth has been astonishing. This growth can be assigned to the necessity that people have to access information on the internet, check their email or use social networks [6] in any place they are. Basically your smartphone is a small computer, a camera, a cellphone, a notebook, a GPS and much more, and it fits in the pocket (most of the time).

Despite how big this market has become, it is also one of the most competitive and prone to change market in the world. Since 2008 the major market holders in this field have changed drastically. At that time Android was just starting to make his first steps with his first smartphone, the T-Mobile G1 [7], while Mac OS X accounted for 8,1% and Symbian for 52,4% of the smartphones sold that year [4]. As seen in Figure 2-1, currently things are very different and in 2012 68,8% of the smartphones market share belonged to Android, 18,8% to iOS and only 3,3% to Symbian [3].

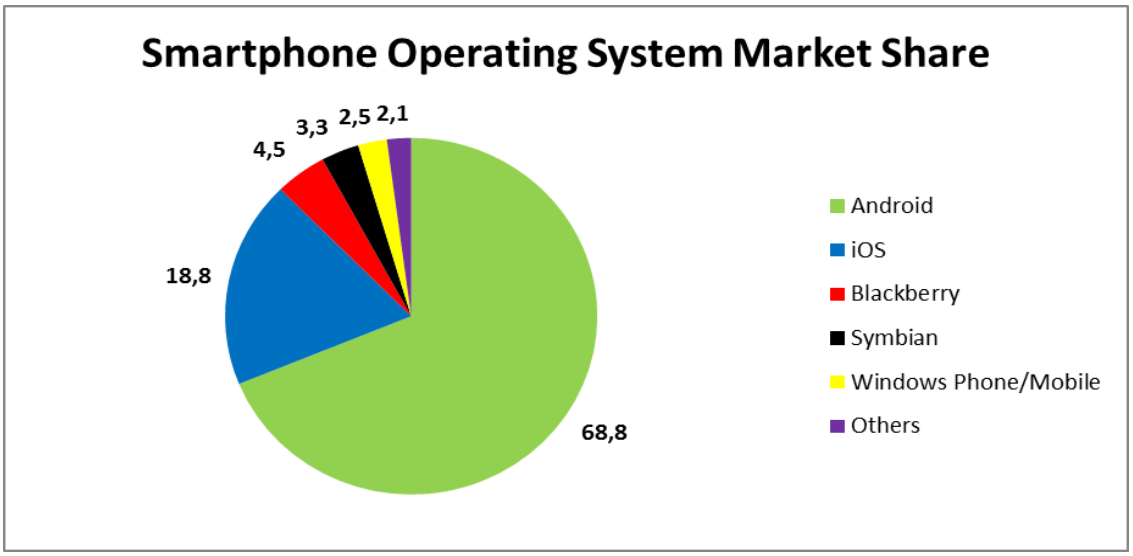

Figure 2-1 Smartphone Operating System Market Share

For smartphone application developers this means that theirs applications will reach more costumers if they have their applications published on the Play Store (Android) and on the App Store (iOS). But this can be somewhat difficult to achieve, since developing an application for Android is very different from developing an application for iOS. These differences start right from the programming language, since Android applications are written in Java and iOS applications are written in Objective-C. This reason alone is enough to make things very complicated, since the application made in one platform will have very little to no code that could be used in the other platform, meaning that the developers would have to make the same application twice.

For this reason developing a cross-platform mobile application using web technologies could bring the solution to this problem, since the idea behind these types of applications is to "Develop once, deploy everywhere". Applying this concept is possible because all smartphone have one characteristic in common: every smartphone has a web browser.

2.2 Application Development for Mobile Devices

With the market mostly dominated, at the moment, by Android and iOS, in order to reach a bigger audience, the developers will have to cater to the needs of the users of this two platforms, but at the same time they want to create rich and well-designed applications for their users, that ideally will look and feel the same on several platforms.

When starting the development process the user will be faced with some very important questions like: what is the main goal of the application? What audience will the application target? Which key features will the application need to answer to the consumer needs? Which smartphone features the application will need to use? Does the application really need to be able to run in more than one platform?

All these questions, specially the last two, will have a great influence in the developer decision of which technology he will use to develop its application. At the moment developers have three choices: go full native; go with web based technologies or use a hybrid implementation. This decision will have a deep impact in the development process, so it is in the developer best interest to have as much information as possible

about each method before taking the decision. Therefore, the next sections will explain some of the key characteristics about each development process.

2.2.1 Native Applications

The most “obvious” approach, when start developing for mobile devices, is to go with the native technology because it’s the one with the most support from the platform developer and allows the mobile applications developers to have more control over the application.

Using native technology means the developer will have to use the programming language and tools required by the platform developers, but on the up side, will have access to all the functionalities of the smartphone (camera, GPS, accelerometer, etc.). In theory, this means that the application will have a better performance and can be richer in terms of features it offers. This is very noticeable in applications that use a lot of complex animations and have a very reach UI, since the programming language used in native development is compiled contrary to JavaScript which is an interpreted language, what make the application a bit slower, since the code is interpreted in runtime.

Another advantage about using a native implementation is the fact that the support from the platforms developers will be greater. This is very important because when a platform developer releases a new version of their OS, it will immediately give the tools to the application developers to start creating applications compatible with the new version, contrary to JavaScript and HTML 5 frameworks that need to catch up, to give their developers what they need to develop to the new version.

However developing in a native environment can have its problems and the main problem that might arise is that all the platforms use different programming language, concepts and development tools. To have an application deployed in several platforms, the developers need to learn more programming languages, work with several development environments and cater to many different devices specifications (something that in Android will happen in any case). This issue will make it harder to developers/companies to have theirs applications deployed in several platforms, because the development process will take more time and resources. Table 2-1 illustrate the variety of programming

languages necessary to develop to each of 5 of the main platforms, the development environment used and the devices each applications will target.

Table 2-1 Mobile phone development technologies

Mobile Platform	*Programming Language*	*Development Environment*	*Devices*
Android	Java	Eclipse	Multiple Devices
IOS	Objective-C	Xcode	iPhones, iPads, iPods
Windows Mobile	C#	Visual Studio	Multiple Devices
Symbian	C++	Multiple Choices	Multiples Devices
RIM	Java	Eclipse	Blackberries

The lack of flexibility by the platform developers in the tools that developers can use to make applications can push away very talented programmers from the mobile world. This is especially true for web developers, since they already have much knowledge with concepts that are very important to mobile applications like usability and user experience, but sometimes don't want to go through the process of learning the details needed to make an efficient and beautiful mobile application.

2.2.2 Mobile Web applications

Despite the fact that developing to all platforms is different in some of their key features and in what they can offer to users and developers, they all have one thing in common: they all have a browser that can be used to access the internet. Developers can take advantage of this and develop their mobile applications, which will run on the browser, using web technologies like HTML5, CSS3 and JavaScript.

A mobile web application is an application that will exclusively run on the smartphone browser. This means that the user have to go to the application URL to use it, which makes them accessible to everyone regardless of the smartphone and platform they are using. This is a good way to reach more users, because it makes theirs applications independent of the platform they are using. However mobile web applications developers

still need to have in mind the different screen sizes that the smartphones can have and the fact that the user can access their application from different browser. The different browser usage can be a problem, because they will have different limitations regarding the support for some of HTML 5 features. However, as seen in Figure 2-2, since more than 50% (54,9% to be more precise) of the smartphone users access the web, either with the standard Android browser or the standard iOS browser [8], this issue is manageable, because both of them use WebKit as their web browser engine, that is what render the webpage.

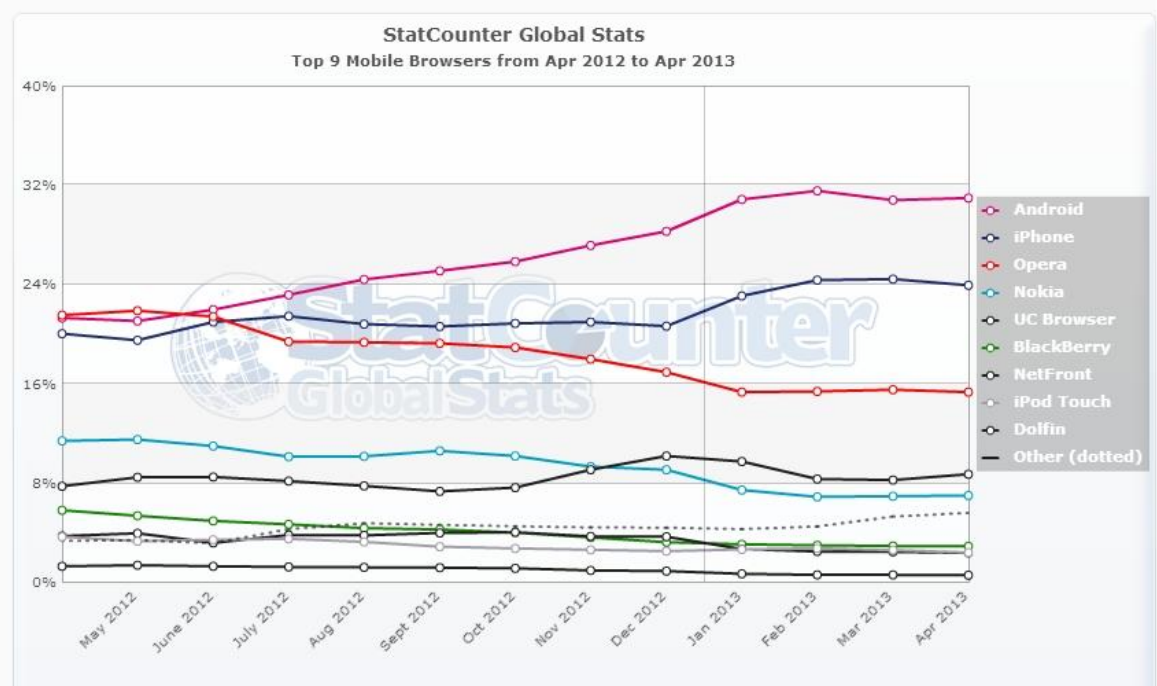

Figure 2-2 - Mobile Browser market Share [8]

Since these applications are accessed through the browser, there is no need to download them from application web stores. Because the applications will be on the web, updates made to them are immediately available to anyone that uses them. On the other side, the application will only work if the user has internet connection (or have a version of the application in cache). Another problem with this is the fact that developers will lose one way to monetize their application, since selling them on the market is not an option.

As for access to smartphone features, web application are a little limited, however this problem has been decreasing in recent times because HTML5 gives some support to these features and in the future the support tends to grow.

2.2.3 Hybrid Applications

Hybrid application is the result of the combination of the two previous development methods. These applications will use web technologies like HTML5, CSS3 and JavaScript for the development, but run in a native container [9], that will make the application seem native. This approach tries to solve the problems that both of the above implementations methods have, while maintaining their main advantages. Since the application will run on a native container, it will be able to run offline and will be able to access to some of the smartphone functionalities. On the other hand since it uses web technologies it will be able to run on more platforms.

This type of applications take advantage of the fact that every platform has a web browser that can be accessed programmatically and use it to run the application (although the user will not be aware that the application is running on the browser). This feature is what let the application run in every platform and although the code will have to be slightly modified according to the platform targeted most of it will remain the same. As these applications use standard web technologies in their development process, this can open the doors to some web developers to enter the mobile application market. The reason for this transition is that they will have to learn a minimal part of native programming in order to package theirs products, making them feel more comfortable during the development process, since they will use tools already know by them.

Like native, hybrid applications can be deployed on the smartphone application stores and like mentioned before this offers the developers another way to monetize theirs application. On the other hand, this will make publishing them more difficult since they have to be accepted by the store quality patterns. Also having the application on the smartphone application store will make the update process to not be centralized, as the user will have to download every new update that the developer launches.

Since these applications run in a native container, they will have easier time accessing smartphone features especially using a framework like PhoneGap to access some extra functionality. This can expand what the developer can do with their applications, making them closer to what a native development can offer.

Like the pure web based application, also the hybrid applications suffer from JavaScript being an interpreted language, making the application sometimes slower than

native applications. However, some good practices can be used to make the user not aware of this problem or at least minimize it. These techniques will be explained in details further in the dissertation. In the next figure, Figure 2-3, it is represented the main characteristics of native, hybrid and pure HTML5 smartphone applications.

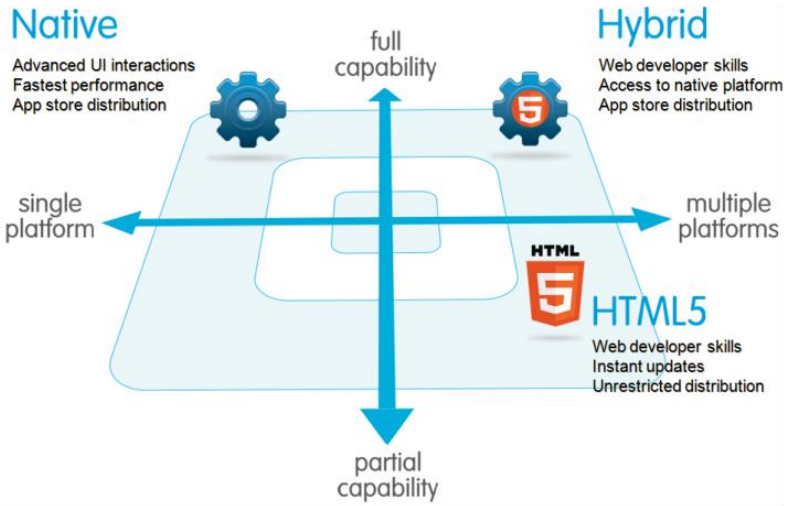

Figure 2-3 Development methods key features [10]

2.3 Web Technologies

2.3.1 HTML 5

One of the main reasons that enable the creation of applications which use web technologies at their core is the evolution they have gone through in the last few years, especially since the beginning of the development of the HTML 5 standard, in 2004. This standard started to being developed by, the now called, WHATWG and W3C joined the development team, in 2006. These two groups worked together until 2011, year they decided to go in different directions, since they had different vision for the project: W3C wanted to release definitive standard and for that reason in December 2012 HTML 5 became a W3C Candidate Recommendation [11]; WHATWG wanted to keep HTML 5 as a "Living Standard", this mean that it will never be "finished" and will continuously be updated [12].

HTML5 offers a lot of features that makes developing mobile applications with HTML5, JavaScript and CSS viable to the point that, according to a study made by Developer Economics, 50% of the developers inquired by their study, already use HTML as a development or deploy technology [2]. This can be attributed to web developers already having the knowledge necessary to work with these technologies, making their interest in the rising smartphone application market grow.

HTML5 is trying to bring innovation to the web. This innovation comes on the back of the flagship features of this technology. These features are meant to make web safer, faster, easier to use in all kind of devices and much more. To accomplish this, HTML5 count with:

- Offline storage;
- Improved Storage;
- Better connectivity;
- Improved file access;
- Richer Semantic support;
- Audio and Video support;
- 3D/Graphics;
- CSS3 support;
- Better performances;
- Smaller improvements that can make the differences;

All the new or improved functionalities that HTML5 has to offer are represented through an icon, making them easy to recognize, as seen in Figure 2-4.

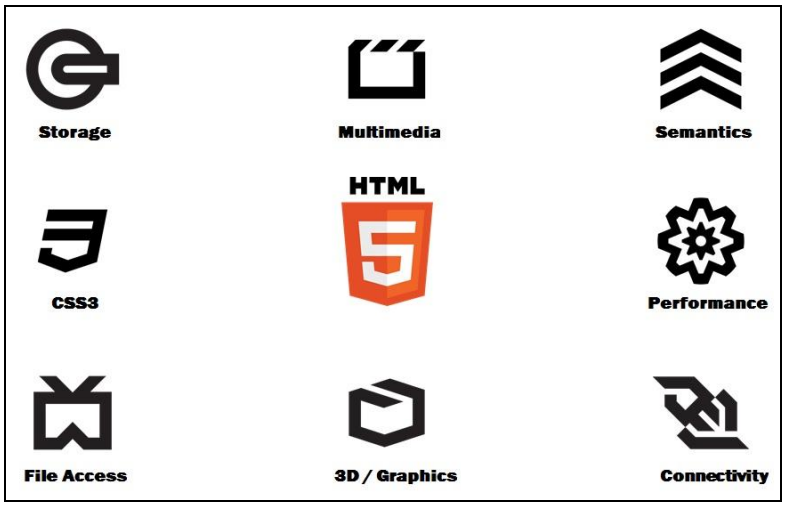

Figure 2-4 HTML 5 features

Most of these features greatly help in developing smartphone applications that can match what native applications have to offer. These features are very easy to use, since HTML5 offers a lot of API's that give the developers a lot of tools to start developing with this technology. Since this dissertation focuses on smartphone application development, some of the most important API's in this field are the ones which use GeoLocation and motions sensors because they take advantage of the smartphone sensors and connectivity, and the media, client side storage and the canvas drawing API, because these ones allow the applications to be more flexible and to include many features the users are already familiar with.

2.3.2 CSS3

Similarly to HTML, CSS also has seen an evolution over the years, which helped the developers create better applications for their users. CSS3, which is supported by HTML5, gives the developers a lot of possibilities in terms of theming their applications to look more polished, beautiful and faster, with the access to new features.

CSS3 is split in to modules that add extra features to the CSS 2.1 standard. Some of these modules are [13]:

- Selectors;
- Box Models;
- Background and Borders;
- 2D and 3D transformations;
- Animations;

From the mentioned modules above the animations and 3D transformation modules are very important in mobile development, because they give the developers the "power" to create more beautiful transition and UI, which will give the user a more pleasant experience.

Since CSS 3 is backwards compatible, if a browser doesn't support a module from the most recent version, it will fall back on the previous version of CSS, but because the most used browsers already give support to many of the features, the developers can take advantage of this to make applications with a more enjoyable interface. Other important detail about the backward compatibility is the fact that some features can be deactivated to

create faster transitions and a smoother navigation on slower devices. This sometimes will be necessary because some of these features can be quite heavy and make the device performance drop significantly.

2.4 JavaScript frameworks for mobile development

JavaScript frameworks are a very important tool to help in the development of mobile web applications. These frameworks facilitate the development by giving the developers features that help them create mobile applications that feel as native as possible, something very important to attract the users to their product. These features include the capability to add touch listeners, theme tweaking, access some device hardware resources, etc.

There are several JavaScript frameworks focused in mobile development such as: jQuery Mobile [14], Sencha Touch [15], Dojo Mobile [16] , Zepto [17], XUI [18], Wink Toolkit [19], etc. In this dissertation the focus is on the first two.

One of the reasons behind choosing jQuery Mobile is that it is based on the jQuery JavaScript framework that is used by a lot of web developers, making it easy for them to make the transition to the development of mobile web applications. Another reason to why jQuery Mobile was chosen is the fact that in the development community it is widely used and though as a good framework, with a lot of nice features, support for almost every platform and can be easily themed to fit the design needs that a developer may have.

As for Sencha Touch one of the main reasons that led to choosing this framework was the hype generated around the Fastbook [20]. Fastbook is an implementation of the web application of the Facebook (that currently is developed using native tools, after abandoning the development with HTML5) with the Sencha Touch framework and their developers claim that Fastbook is much faster than the original implementation. That claim generated a lot of curiosity around Sencha Touch and this work will try to see if they are indeed true. This reason alone is not enough to choose this framework, but ST is also very production ready (have tools which make the development process smoother), have a solid MVC architecture and since is based on Ext JS, some web developers who already use this framework, could make a smooth transition to the mobile application development world.

2.4.1 jQuery Mobile

jQuery Mobile is a JavaScript framework that has at its core jQuery and jQuery UI, but has implemented many functionalities which makes it suitable to build mobile applications that use touch as the main way of interaction between the user and the device [14].

JQM first version (1.0 alpha) was released in October 16th, 2010 [21] and at the moment of the writing of this dissertation, the stable version is 1.3.1 [22]. jQuery Mobile is under the MIT License [23] and it's sponsored by the Mozilla Corporation, Palm, BlackBerry, Nokia, DeviceAtlas, Jive, dotMobi, rhoMobile and Filament group.[14]

One of the goals of jQuery Mobile is to give support to as many platforms as possible. To understand how the platforms behave with the JQM framework, they use a grade system, that will grade the platform with A (full support), B (full support except for Ajax) or C (basic HTML) according to which features they can use [24]. Some of the main platforms that jQuery Mobile attributed "A" grade are:

- Apple iOS: 3.2 to 6.1;
- Android: 2.1 to 2.3, 3.2, 4.1 and 4.2;
- Windows Phone: 7.5 to 7.8;
- Blackberry: 6 to 10;
- Palm WebOS: 1.4 to 3.0 [25].

To set up JQM the developer needs three files that must be included in the index.html (his main web page file): the jQuery core JavaScript file; the jQuery Mobile core JavaScript file, which include all the touch related features; the jQuery Mobile core CSS file, which has the default theme to make the UI element have a mobile appearance [26].

As said before, JQM has some features that, in general, are attractive to the development community and make it a widely used framework. These features are focused in making the applications as lightweight as possible, support a great amount of platforms and be easy to learn, for both new developers and for web developers who already have worked with the jQuery framework [27]. To achieve these objectives JQM offers a framework built on jQuery core, that support a HTML 5 markup driven architecture to give

developers an easier learning experience, many responsive design technics that make the process of developing for multiple platforms and screens sizes simpler, support for mouse and touch events to give even more support for different platforms, modular architecture so the developer can choose what he want to include in its application, thus making it lighter and unified UI widget that is easy to customize which make the development for the different platforms similar [28].

Another tool that JQM offer so the developers can customize their applications to their needs is the ThemeRoller [29]. This website gives the possibility to the developers to customize their own theme, defining text font, UI elements colors, shadows, etc. up to 26 different swatches and after all the theming process is finished, the developer need to download the CSS that the ThemeRoller generates [30]. Other interesting feature of this tool is the possibility to upload other themes that the developers created earlier and make the changes needed. Finally ThemeRoller let you choose the version of JQM that the developer is using, so the generated CSS file has only features supported by that version, thus avoiding compatibility issues [29].

Although JQM tries to give new users an easy time when learning the tricks about the framework through its HTML 5 markup-driver architecture, this framework offers the possibility to tweak many aspect about the mobile application when the developers goes deeper into the framework. Through the API that is given, the developers can control touch, mouse, scrolling, layout, animation and several page related events (load, transition, initialize, remove, etc.), orientation changes, etc. [31] and then apply several methods (like changing the current page or show a loading message), also provided by the API [32], to better control how the application behave, giving the developer a much larger array of possibilities when developing it. To have access to all these methods and events the developer have to use the object '$.' for jQuery ones and '$.mobile' for jQuery Mobile specifics ones [31], [32].

When defining the interface of a web application, which is being developed using JQM, the developers will mainly write HTML 5 with some CSS code to theme their interface. To have access to the widgets that JQM mobile has to offer, the developer can use one HTML 5 feature which JQM takes advantage that are the custom data attributes. This HTML 5 feature let JQM define custom attribute in the HTML 5 tags, which the JQM then uses to initialize and configure their widget. The JQM framework uses the attribute

"role" to define what widget is called and depending of the "role" the component will have different attributes. "Theme" is one of the most common attribute, that calls one of the swatches defined in the CSS theme file [33]. The following code snippet displays how to declare a button component with the theme named 'a', which is defined in the jQuery Mobile CSS file.

```
[<a data-role='button' data-theme='a'>Button</a>,
```

2.4.2 Sencha Touch

Sencha Touch is a HTML 5 mobile application framework, developed by Sencha, which has in its core the class system from Ext-JS 4 and uses the MVC architecture [15], [34].

The Sencha Touch project was announced on June 14, 2010 with the company Ext JS changing its name to Sencha and announcing a collaboration with JQTouch and Raphaël [35]. On November 15, 2010 Sencha released Sencha Touch 1.0, five month after the release of their beta version. On that same announcement, Sencha stated that Sencha Touch would have a free commercial license [36] and at the moment of the writing of this dissertation Sencha Touch is under the free commercial license and open source license (GNU GPL license v3) for application development and have a paid commercial license for OEMs [37]. The current stable version (and the version used in this work) is Sencha Touch 2.2 that was released on April 15, 2013 [38].

Although not giving support to as many platforms as jQuery Mobiles does, Sencha Touch still gives support to the most import platforms on the market [34], which include:

- iOS 4+;
- Android 2.3+ and 4.0.3+;
- BB OS 6 and 10;
- BB Tablet OS 1;
- Windows Phone 8;
- Kindle OS 6.2.2,

As said above, ST uses in its core the Ext JS 4 class system. This class system offer the developers a kind of class-based programming that let them use techniques like inheritance, dependency loading, use many configuration options (a good tool to tweak the applications),etc. [39]. The dependency loading is an excellent way to make the applications smaller, only using the framework resources that are needed, resulting in a smaller and lighter application.

Another very important feature that ST presents is its MVC pattern. The idea behind MVC pattern is to separate the business logic (Controllers and Models) from the user interface (the Views) of an application. To be more specific the View contains the information about the UI, the Controllers contains the information about how to handle the user actions and events and the Model defines the data/business model used by the application [40]. In Sencha Touch this is very useful for three main reasons [41]:

- History support: the possibility to navigate within your application, according to your navigation history;
- Deep linking: possibility of accessing any screen in your application;
- Device Profiling: since developing cross-platform applications is one of the reasons to use a tool like Sencha Touch, the ability to make profile for the different platform, devices and even screen sizes is invaluable.

To start developing with Sencha Touch the developers will have to download the Sencha Touch SDK and the Sencha Cmd from the Sencha website [42]. One thing to notice is that to download the Sencha Touch SDK the developer will have to provide an email. The Sencha Touch SDK contains all the files needed to start the development. Sencha Cmd is a command line tool, that enables the user to easily do all tasks related to the development life-cycle of the application (create project, build, deploy, etc.) [43]. The Sencha Cmd needs to be installed, in order the developer to have access to their functionalities through the command line.

After having downloaded the necessary resources, the developers can use the Sencha Cmd to generate a Sencha Touch application. Something that is worth noticing is that when creating the application with the Sencha Cmd, the developers need to run the create command from the folder of the Sencha Touch SDK (in the command line), so the Sencha Cmd knows which SDK will be imported to the newly created application [43]. When the project is created, it has the following structure [44]:

- .sencha: folder containing some specific Sencha files that are mainly used to configure how Sencha Cmd will build the application;
- touch: folder with all the ST resources, from the source to resources files.
- app: folder where all the source code of the project will be. This project already has the MVC structure (controller, model, view, store, profiles folders). The first view of the project, which is in the file Main.js is automatically created and placed in the view folder.
- resources: folder containing all the CSS and SASS files, icons, images and the loading screen (for iOS only). This folder contains all the style sheets and the applications visual resources (images and icons). When the application is generated the main.css and the main.scss files are automatically created. Also any exterior data can be put in this folder, as long as it is then referenced in the app.json file;
- app.js: the main JavaScript file of the project that has the project initialization logic;
- app.json: the project configurations file for deploying the application. This file contains information about the JS and CSS files needed to the project, which resources should be imported to the project build, the application namespace, among other smaller configurations;
- packager.json: similar to the app.json file, but the packager.json has the configuration for the native packaging of the project. This file has options like the application name, the platform where the project will be deployed (Android or iOS, both in the device or in the emulator), native SDK local path, applications permission (just for Android), etc;
- index.html: file loaded by the application that will be displayed on the screen. This file is almost all time unchanged, since all the imports are in the app.json file.

This structure, to someone that is not familiarized with the use of the MVC architecture pattern, can, at first, feel a little overwhelming, but after gaining some experience it is very helpful to keep the project very organized and clean.

One import feature about the structure of the project is the presence of SASS files. A SASS file is nothing more than an extension of CSS3, that adds nesting rules, mixins,

variables, selectors inheritance, etc. [45]. With this features that SASS offers, creating a totally custom theme that is based on the ST theme is very easy. However, since SASS is not a styling language but a scripting language, ST uses Compass to compile the SASS file to CSS. Compass is a CSS authoring framework that uses Ruby files to configure its build [46]. Due to this reason, the user needs to have Ruby installed to compile the developed SASS themes. This theming method can be quite useful, since changing the whole theme only requires changing one file and a few variables and if the user still want to make small change to the theme, it's still possible to create a small CSS file that will address the details that the user wants to change.

After setting up all the necessary resources, the developer can use the already created main.js file to make the initial screen of the application. The creation of the interface is entirely developed through the use of JavaScript, more specifically through the ST classes system stated above. To invoke these classes the developer has to use the global namespace Ext followed by which UI component he wants to use. After invoking the component it's possible to configure its attributes, properties, methods and events. Which options each components has can be consulted in the Sencha Touch Docs [47].

Since ST uses the MVC architecture pattern, all the views (screens) the developer wants to use should be placed under the View folder inside the app folder. In the Controller folder the user should put all the code that will handle the actions of the application, that mainly consist of user event handling like tap, scroll, etc. The profile folder offers the possibility to the developer to define profiles for different type of devices where the application will run, which is very useful when working with cross-platform frameworks. The Model and Store folder are closely related: the Model folder has the data objects of the Application; the Store folder has the files that load the data object represented by the files in the Model folder and then can for example fill UI elements like lists [41]. All the files will then be loaded by the app.js when it defines the application with the Ext.application class. In the code snippet below, it is an example of an application named "Test" that consist of three views (Main, Login and Menu), three controllers (one associated to each view), two Profiles (Android and iOS), two Models (Client and Seller) and two Stores (Clients and Sellers). After the files have been loaded, the launch function is invoked and the Main view created. The source files of the project can have the same name if they are in different folders of the MVC structure.

```
Ext.application({
        name: "Test",
        models: ["Client", "Seller"],
        stores: ["Clients", "Sellers"],
        views: ["Main", "Login", "Menu"],
        profiles: ["Android", "iOS"],
        controllers: ["Main", "Login", "Menu"],
        launch : function()
        {
                Ext.create("Test.view.Main");
        }
});
```

2.5 Mobile Development tools - Phonegap

When talking about web based mobile application a topic that is much discussed is the use of Phonegap. Phonegap is an open source framework used as a container technology to web applications, so they can be installed as native applications [48].

Phonegap is widely used because it offers an interesting set of APIs which give access to some smartphone features and because it make the process of deploying application in several platforms easier. However using Phonegap has its problems and the main one is that using Phonegap will make the application heavier since it will put another layer of code in the application. Since Phonegap most of the times is used in conjunction with another mobile development framework (like jQuery Mobile or Sencha Touch), this can make the application slower.

In simple applications the use of Phonegap can be somehow replaced by simply launching the code developed through the smartphone browser from inside the application. In the case study that will be discussed next, this was considered the more appropriate approach since the application doesn't need to use many smartphone resources and to study performance avoiding any additional layers of code was preferable.

2.6 Conclusions

The current development of both the smartphone market and web technologies is opening many opportunities in the development of mobile applications. Developers will have a harder time choosing which technology to use, but at the same time they will have a lot more tools to facilitate the development process. Furthermore, with the growth of the mobile market, developers will have more options to what they want to focus with their applications.

Web technologies are evolving every day and the features they offer are growing, allowing for more complex, faster and more beautiful application. However, developers must consider who their target audience is and what they want to accomplish with their applications before start developing, since, despite the improvements in web technologies, sometimes they are not enough.

The following chapters will try to give some insight about web technologies and what they can accomplish when developing smartphone applications.

3. The Self-Care Mobile Application

3.1 Introduction

The first step taken in this study was to develop a mobile application that was developed using native technology and then it was replicated resorting to web technologies. This approach was considered the best one to take, since replicating something already implemented avoid a lot of phases in the development process (choosing concept, UI drafts) that are not very important for this study.

The program that was used has a case study of the web mobile frameworks was Self-Care. The objective of this mobile application was to enable its users to consult and change some information about their account which is used to track events, promotions and service usage of your mobile phone.

Although the version that was used as model was incomplete, it had all the main functionalities implemented. These functionalities are:

- Pseudo Log In: the login was not fully implemented, since the password was not necessary to access the system, but to see the information from the user, the user ID had to be known to make a request to the server (in this dissertation, it's not important that the log in was not implemented since it was server side functionality that the mobile application would only access).
- Check account information: the application allowed the user to check the number of MMS, SMS, calls (and their total time) and Internet data consumed. Was also possible to see detailed information from each of these types of events. Furthermore the user could check his balance from the different services provided.
- Recharge the account: with this application is possible to recharge the user account using a specific code.
- Consult and change the user mobile phone plan: the user can see which is the current mobile phone plan he is using; the list of all the available plans;

check information about each one of them; change to another one if they want to.

- Subscribe or unsubscribe services: the user can view a list of the services he had subscribed and the services available to him. For each of these services is possible to see detailed information and subscribe to the services that are available.
- Check Promotions: see what promotions are available to the user.

Analyzing the aforementioned features carefully, it's easy to conclude that this application will mainly consume web-services and display their result to the user. The changes to the user account will also be done through web-services offered by the server. This type of the applications, that only have simple functionalities, are a good way to study new technologies before jumping into something that can be overwhelming for someone who want to start developing smartphone hybrid applications.

Beside the actual functionalities, it is also very important for in this dissertation to try to replicate the interface that was implemented using native language. The Self-Care interface is very simple and minimal, which again make this application a good example when trying to start using new tools and technologies, and study the UI elements that the JavaScript frameworks offer the developer.

3.2 Interface and Functionalities

The interface and the functionalities of a smartphone have to blend very well together, to give the user the best possible experience. To have a better understanding of what was done when replicating this native application with Web technologies, in the following section, every screen and their functionality will be explained.

3.2.1 Log In

The first screen that is presented to the user when he starts the application is the **Log In** screen. This screen has to be very simple so the user can easily input his credentials and start using the application. In order to have the simplest interface possible, it's desirable to have the least amount of UI elements necessary, so the screen will mainly need

two textboxes (one for User ID, one for password) and a button to execute the login procedure. The pretended interface, as presented in the original Android application, can be seen in the Figure 3-2.

For the login functionality of this page, handled on the server side, the mobile application only has to access the web service the server provides to authenticate the user. To access the web service the mobile application has to make a HTTP REST request, with GET method, which is triggered when the user tap the button. Since the Login functionality, on the server side, was not fully functional, to give access to the account information and methods, the mobile application only had to request the user information and to accomplish that it was only necessary to provide the User ID to the web service. The ID is stated in the URL used to make the request. If the request is successful the user will be redirected to the menu screen, otherwise the user will be informed of the failed login. All this process is presented in the Sequence diagram in Figure 3-1.

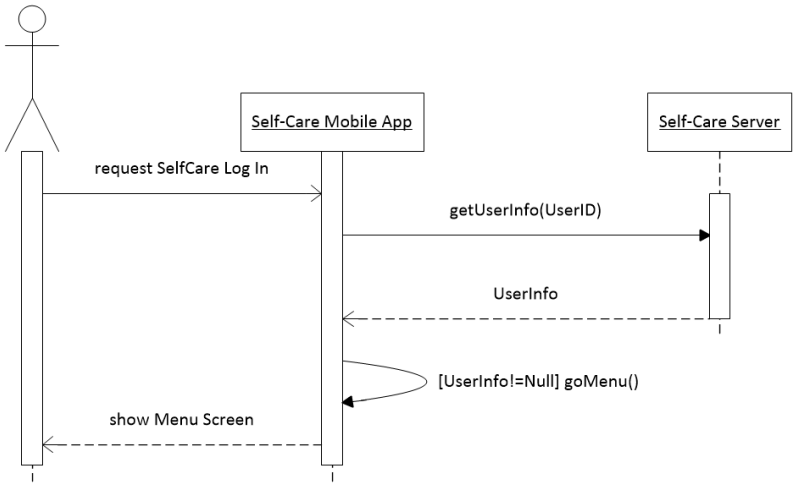

Figure 3-1 Sequence diagram of the login process

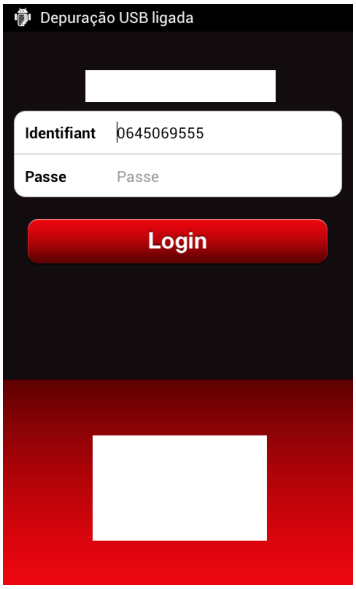

Figure 3-2 SelfCare Login View in Android Native

3.2.2 Menu

When the log in is successful the user is redirected to the menu screen. This screen gives access to all the features of the mobile applications, through a set of buttons. The original Self-Care Android interface is presented in the Figure 3-3.

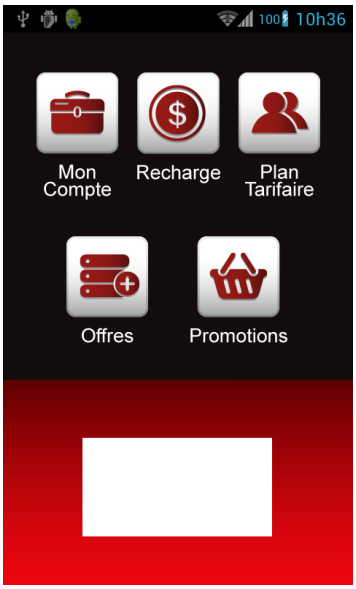

Figure 3-3 SelfCare Menu View in Android Native

3.2.3 Account

The account screen displays the information about the events the user performed and balance from the user account. The information about the events performed is grouped by type (SMS, MMS, voice calls and Internet access) and their total is displayed on a list. Also this screen will display a list of current balance the user has for each service used.

To get the information that will be displayed to the user, the mobile application has to access the server. The information is retrieved through a HTTP REST request, with GET method. The two block of information displayed to the user are returned by different web services, therefore the mobile application will have to make two requests to the server. The first request will get all events from the user and the second a list of the balances for each service. The information returned by the server is in JSON format. The process described above is illustrated in the Sequence diagram in the Figure 3-4.

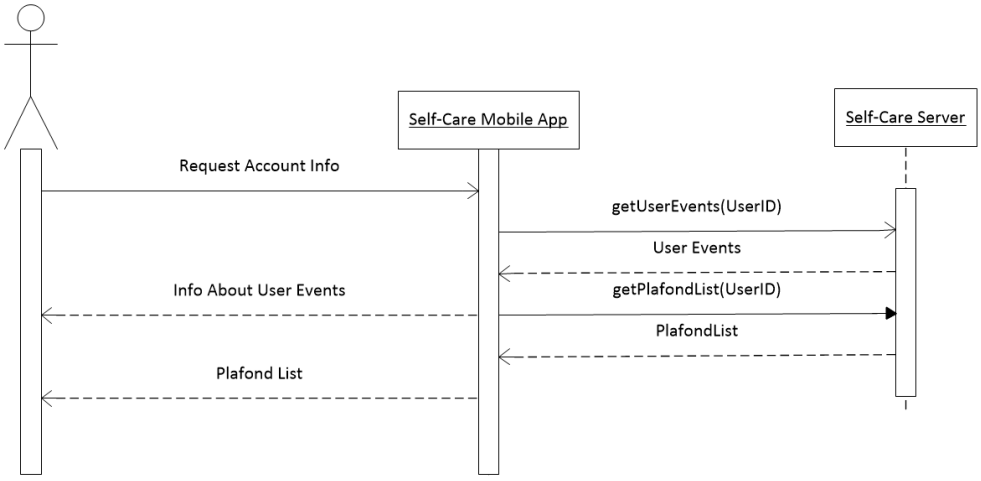

Figure 3-4 Sequence diagram representing the request made in the Account screen

One important detail to have in mind is that the events can be consulted by type, this mean that this screen should give access to a more detailed view of the events of a each type.

This screen will have a navigation style UI, with a bar in the top of the screen that will enable the user to access other options, which is presented in the Figure 3-5.

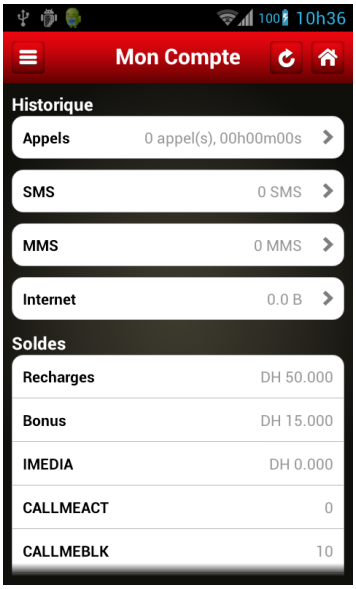

Figure 3-5 SelfCare Account View in Android Native

3.2.4 Recharge

The Recharge screen is fairly simple, since the only features it offer is the possibility to recharge the account of the user. The screen will need a text box where the user will put a code that will recharge the account and a button to issue the recharge function. The interface which was replicated is displayed in the Figure 3-7, which presents the original Android application Recharge page.

In the Figure 3-6 it is possible to observer that to perform the recharge function, the mobile application access a web service on the server through a HTTP REST request, with a POST method. The code is verified on the server and the result of the verification is returned to the mobile application. The user is then informed on the result of the request.

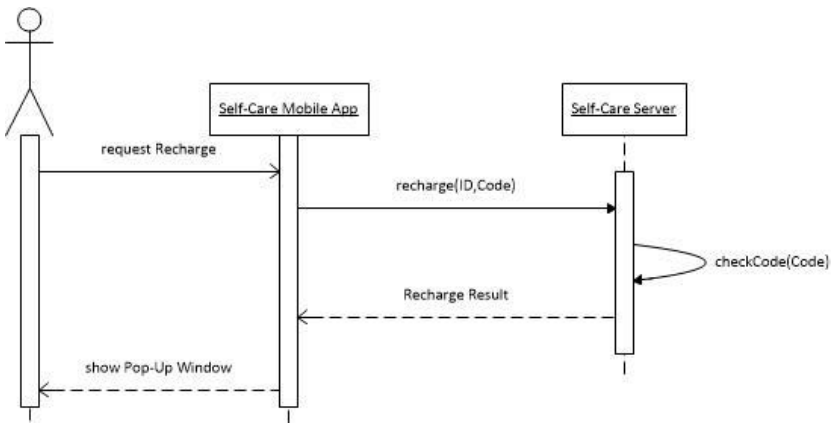

Figure 3-6 Sequence Diagram of the recharge process

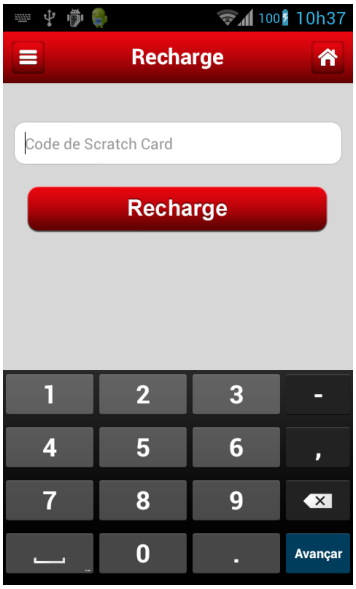

Figure 3-7 SelfCare Recharge View in Android Native

3.2.5 Mobile Plan

This screen displays the mobile phone plans that are available to the user, including the one being used at the moment. Since these mobile phone plans are represented only by their name, a list is used to display them to the user, with the current mobile phone plan being highlighted. All the mobile phone plans have details that are not shown at first to the user, but can be consulted if the user taps on the list item. The interface described for the Mobile Plan page is displayed in Figure 3-10, which was retrieved from the native Android application.

To populate the list with the info to display the user, the Self-Care mobile application has to make two REST requests, GET method, to the server: one requesting the user information's (because the mobile phone plan of the user is one field of his general information); other requesting the list of all available plans. When the data is retrieved from the server, besides the name, which is necessary to display in the list, all the other data is saved, so it can be presented, if the user want to access details about the plan. Since the server returns the information in JSON format, it needs to be parsed, before it can be saved. The Figure 3-8 represents this process through a Sequence Diagram.

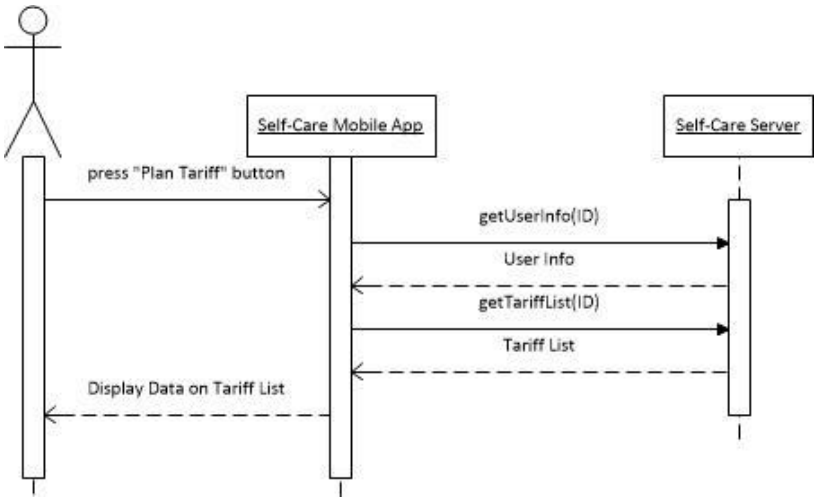

Figure 3-8 Sequence diagram of the process for the retrieval of mobile phone plans

In the detail view it is possible to change the mobile phone plan being used. To accomplish this, the mobile phone application does a HTTP REST request, POST method, to the server with the User and the mobile plan ID. The server verifies if it's possible to change plan and returns the outcome of the request to the mobile application, which in turn is displayed to the user. This process is also explained in the sequence diagram of the Figure 3-9.

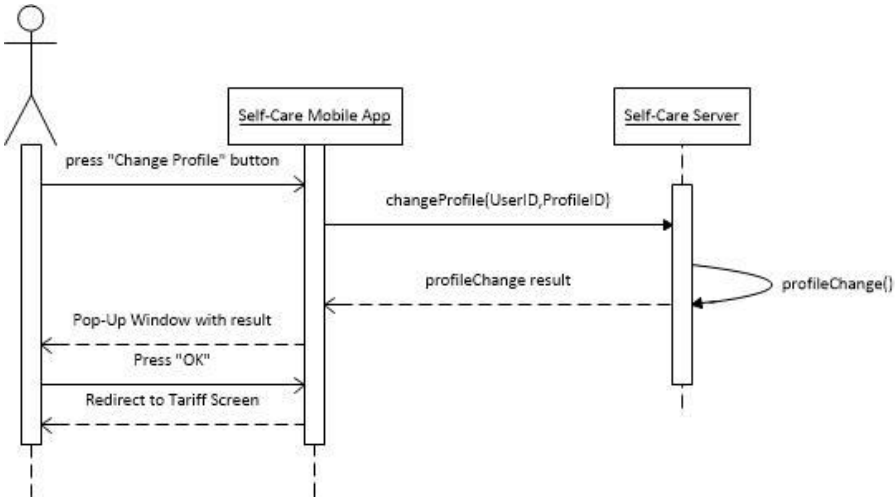

Figure 3-9 Sequence Diagram of the process to change mobile phone plan

Figure 3-10 SelfCare Mobile Plan View in Android Native

3.2.6 Services

Other functionality available to the user is to see what services he can use and subscribe to them. Besides the services available, this screen also shows the user the services that he already has subscribed.

Since there are two types of information in the screen there must be a way to set them apart. This is done by placing an UI element (a button, a switch) on the screen, which will control the information displayed on a list with the name of the service. The interface that will be replicated is presented in Figure 3-13.

In order to get the data that will fill the list, mobile application must make two HTTP REST request, GET method, to the server: one to get the services subscribed; and another to get the services available (unsubscribed). The data received from the server is in JSON format and therefore must be parsed before it can be saved and displayed to the user. The process is illustrated below in the Figure 3-11.

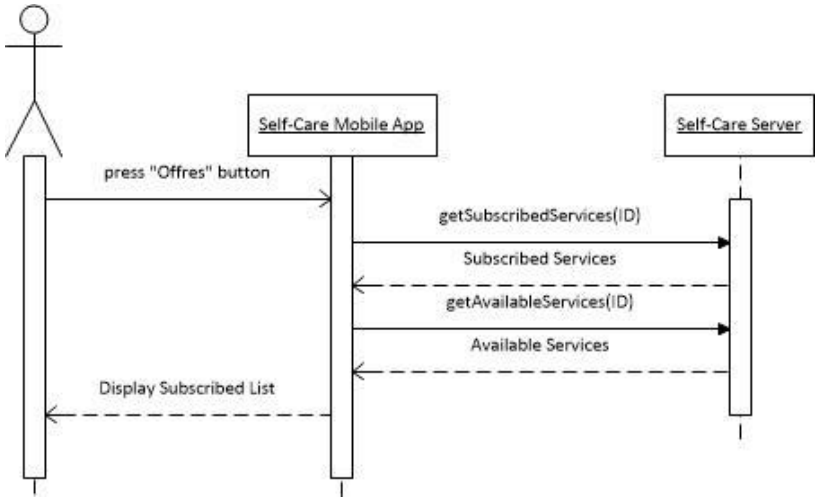

Figure 3-11 Sequence diagram of the process for the retrieval of the Services

This screen also offers the possibility to the user to access details from each service. In the details screen, if the service selected is unsubscribed, there will be an option to subscribe to it. This process is very similar to the process of changing mobile plan, since this is also done through a HTTP REST request, POST method to server, in which is specified the User ID and the Service ID. The server will process the request and return to the mobile application its result. All the request and responses from the server are represented in the sequence diagram in the Figure 3-12.

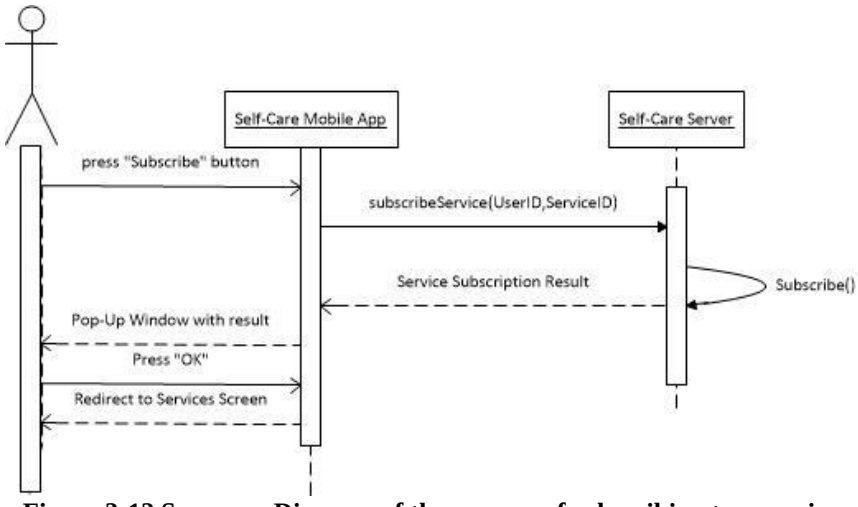

Figure 3-12 Sequence Diagram of the process of subscribing to a service

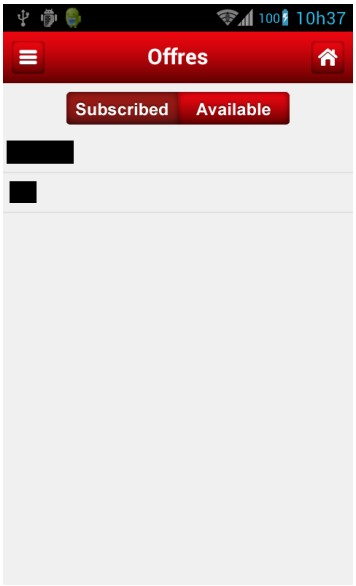

Figure 3-13 SelfCare Services View in Android Native

3.2.7 Side Menu

One common thing to all the screens (except the Login and Menu screen) is the presence of a side menu. Although this side menu is not visible at first, it's an important interface element and was very important to explore the possibilities of the frameworks studied, since it is something not trivial to implement, unlike the other UI elements.

As said above, this side menu is not visible initially. To access the menu the user has two options: the user can drag the screen from the left to the right to display the menu; or the user can tap on the menu button in the top toolbar. This menu contains five buttons which give access to the option presented in the Menu screen. The original application presented a very interesting interface, which is displayed in Figure 3-14.

The interesting characteristic about this is not the menu itself, but the swipe/drag component, offered to the user, to activate the menu. This feature would be very important to duplicate because mobile applications must respond to many user triggered events and this is a good example of that.

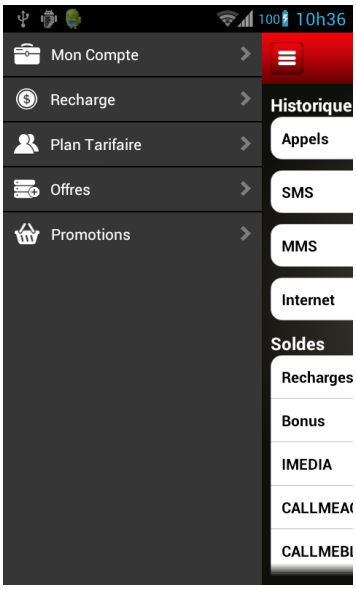

Figure 3-14 SelfCare Side Menu in Android Native

4. Self-Care web based application

With all the interfaces and functionalities from the native application well defined, the next step in this study was to develop Self-Care with the HTML5 frameworks that were mentioned before. Since the general opinion from the mobile developer community was that JQM was easier to learn than ST, this was the framework chosen to be used first.

In a first phase the application developed was tested and deployed on an Android device, but this didn't have any influence in the development process, since the fact that web technologies are cross-platform was the reason why they were chosen as an alternative to the native development, in the first place.

The approach taken in the development process was, to implement the interface of every page and then implement all the functionalities. This approach was selected since many software development methodologies initially work on mock ups of the interface before advancing to the development of the functionalities. Since this application already has the entire interface designed (the one used in the native applications), the ideal way to adapt to the developments is to start by implementing the UI.

Since there were no details given about the implementation of the application, during the development process the UI elements and resources used were the ones that could better replicate the native application.

4.1 jQuery Mobile

4.1.1 Setup

As said in the introduction about the JQM framework, to start the development process is necessary to make reference, in the index.html file, to the JavaScript's files of the jQuery and jQuery Mobile framework and to the CSS file of the jQuery Mobile. This can be done either by: refereeing to a CDN in the index.html, that host the files, which avoids the need to add the files to the project, saving a small amount of storage space; or by adding the files to the project and then the index.html referring to their local path,

preventing that the application will stop completely its access to the JQM functionalities if it can't reach the CDN. Since the three files have a combined size of only 327 Kbytes and, in this case, being able to test the application without Internet access was desirable, it was selected to include the files in the project folder.

4.1.2 Structure

With a markup driven architecture, putting all the UI elements in the screen, using only HTML5 is the easiest way to create the interface. But once again, a development decision has to be made on how to organize the files in the project. In JQM, the developer can have all the pages in a single HMLT5 file, one HTML5 file for each page or a mixture of the two. Having all the pages in the same file will make the initial load slower but after that the transitions between pages will be faster, since the application doesn't have to fetch the new page on a different file. Having this in mind, with the small amount of pages that the Self-Care application have, it was decided to have all of them on the index.html file.

In JQM to declare a view or page, the developer needs to put an attribute data-role set to 'page' within an element (in this project, it was always used in a element) inside the <body> of the HTML5 file. Due to this and since the option in this work was to use a multi-page template, each view in the index.html file was defined by the use of this attribute and to identify each page when navigating between the divs, each one had a unique identifier. Other detail taken into account when using multi-page template is that the first screen displayed when the application is loaded, is the first element with the data-role "attribute" set to page in the index.html file, which in this case will be the Login screen.

4.1.3 Login

After the decisions about the structure of the application were made, the focus shifted to the development itself. As stated in the Self-Care mobile application description, the first screen of the application is the Login screen. This screen has a title, two labels that

identify the user ID and password textboxes to their right and a Login button below. The background has an image that was also used in the native version of the application.

To set the background image, it's necessary to define the "style" of the with data-role= "page" attribute and put the path to the local image in the field background (in this case the image is stored locally but the path could be to an image hosted online). Also it's important to define the size of the background, but since in this case there were several images supplied for the different screen resolutions (LDPI, MDPI, and HDPI), the size could be set to 100% for the height and 100% for the width, this way the image will fill the entire screen.

To make the label and the respective textbox be aligned side by side, it was used a field container. This JQM element groups the two elements together to help the user visually identify the purpose of the form element they are filling [49]. The label and input elements in the field container are plain HTML elements, with the type of input expected being defined in the type attribute. In the code snippet below, the "Identifiant" field is a number (one of HTML5 new input types [50]) and the "Passe" field is password.

```
<div data-role="fieldcontain">
      <label style="color: white; font-size: 15pt"
            for="username">Identifiant</label>
      <input type="text" name="identifiant"
            id="username" value=""
            placeholder="Username"/>
</div>
```

The button that is on the screen is an anchor HTML element with the data-role attribute set to "button". To theme the button to look like the one in the original application, it was used the ThemeRoller tool. With the ThemeRoller it was possible to open the JQM original CSS file and add more swatches (with a red theme in this case), export the resulting CSS and then include it in index.html. Besides styling the button, also the toolbars were styled, so the next screens already have the swatch they need to emulate the original application. The button doesn't have any onclick() event associated with it because the tap event is handled in the JavaScript file that listen and handle all the UI related events. To listen to these events it's important that the button has a unique identifier, as in the code below.

```
<a href="#" data-role="button" data-theme="f" id="redButton"> Login</a>
```

To set the title, it was used a plain HTML heading (<h2>) aligned to the center of the screen.

When the user taps on the Login Button, the mobile application will start the login process. This process begins with the application "catching" the tap event. JQM offers listeners for several events, especially touch related events. In this case the event that the application was expecting was the "vclick" on the Login button. The "vclick" (virtual click) is an event that can be handled as a tap or a click. This was useful in the initial stages to test some of the application functionalities in the desktop browser. Other good reason to use the "vclick" event is that this event is triggered immediately after the user lifts his finger of the screen. This is very useful on mobile devices, because some touch events will have a 300ms delay before being handled, who makes the application less responsive, thus feeling sluggish.

When the application catches the events, it will show a loading overlay triggered by JQM. The overlay has several configuration options which include the text that is displayed, if the text is visible, the theme of the spinner, etc. The loading overlay, in this case, is configured when the application is launched

The configurations used were the ones, stated in the code snippet below.

```
$.mobile.loader.prototype.options.text = "Loading";
$.mobile.loader.prototype.options.textVisible = true;
$.mobile.loader.prototype.options.theme = "a";
```

An example on how to activate the spinner is displayed in the following code snippet.

```
$.mobile.loading("show");
```

While the overlay is displayed, the login begins. In a login process the user has to put the username and its respective password as expected. But since the login web service was not fully implemented at the time of the development of this application, the application only does a pseudo login. This means that the user will only have to put its ID on the textbox and if that ID exists in the database the user will be logged in. To verify if

the user is registered in the database, the mobile application creates and sends an HTTP REST request with the method GET, to retrieve information about the user. To create and send this request, the jQuery API offers a function to perform HTTP (Ajax) requests. With this function it's possible to configure the URL to which the request is made, the method of the request (GET in this case), the format of the data in the answer (JSON) and more important, the actions to take, in case of success or error, among many other things [51]. To get the user information the URL of the request, must have the ID of the user. Being able to configure the actions to take, in case of success or failure, is very useful, because in this case, if the request succeeds the user will be redirected to the Menu screen and if it fails, a pop up notification message informing the user that the ID and password are incorrect is displayed. To change the current page, it is used the method changePage() from the JQM framework. This method will display the page that has the id passed as the first argument. The second argument is the configuration options of the page change, that can include animation of the transition, data to send to the next page, among other configurations [32].

```
$.ajax({
      type : "GET",
      url : "SERVER URL\USER ID"
      dataType : "json",
      success : function(data, textStatus, jqXHR) {
            $.mobile.changePage("#menu", {});
      },
      error : function(jqXHR, textStatus, errorThrown) {
            $.mobile.loading("hide");
            navigator.notification.alert('Login Failled',
                  null, 'Self Care', 'Ok');
      }
});
```

The full REST request was displayed in the code snippet above, to exemplify how the function works. From now on, only important settings of the request will be shown, like the success function, to avoid unnecessary repetitions.

In figures 4-1 and 4-2 are displayed the resulting interface, when deploying in iOS and Android respectively. Is possible to see that the interface is very similar in the two platforms (as expected) and comparing to the original application, despite the fact that the elements to fill the user ID and password are different, the global look and usability was replicated successfully.

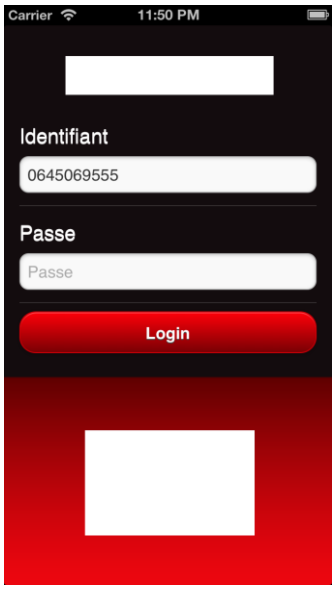

Figure 4-1 Login Screen developed with jQuery Mobile deployed in iOS

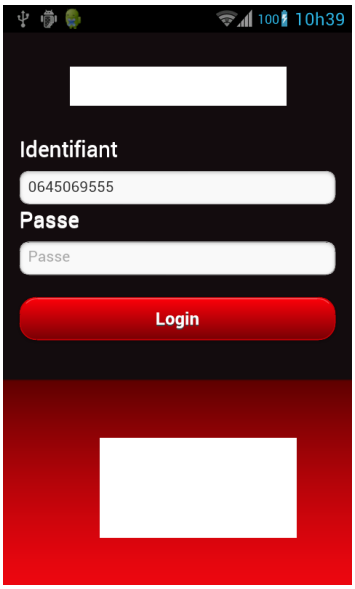

Figure 4-2 Login Screen developed with jQuery Mobile deployed in Android

4.1.4 Menu

The menu page is from where the user has access to all the functionalities of the application. This screen displays to the user five buttons that give access to those functionalities and a background with an image, like in the previous screen. In the Figures 4-3 and 4-4 it's possible to see a representation of the screen described above.

For each of the button in the original application there are two images: one when the button is not being pressed; one for when the button is being pressed. To replicate this with HTML5, each button is a <a> element, which has inside a <img> elements. The images of the button are stored in the application and therefore the access to them is done locally by the "src" configuration in the image element as seen in the code snippet below.

```
<a class=btn1 href="#" id="account">
      <img src="Path to the Image"
            style="margin-right: 3%" id="account_btn"/>
</a>
```

When the button is tapped, the event "vmousedown" is triggered and the source of the button changes to the tapped button image. Changing the source of the image like this can be very useful, because it gives the user the feedback that the button is pressed, giving

him the illusion that application is faster than in reality is. This process is very simples to accomplish with the jQuery framework, since jQuery offer the possibility to easily access DOM elements by their ID.

```
$(account).on('vmousedown', function(event) {
	var account = $("#account_btn");
	var value = account.attr("src");
	value = value.replace("original path", "change_path");
	account.attr("src", value);
});
```

When the user lifts the finger off the screen, the “vclick” event is triggered and the current page changes to the page of the selected option. Depending on the options selected the function that handles the “vclick” event will launch a loading overlay, if the content of that page is not immediately available to the user. Also the function will restore the default image of the button.

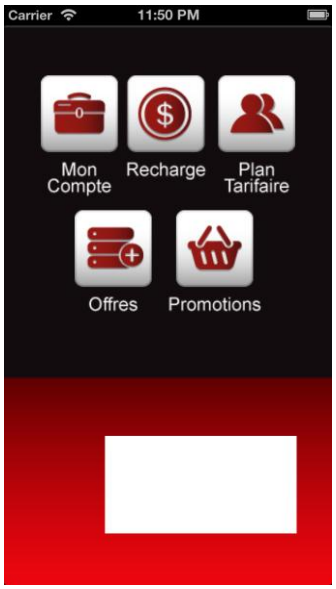

Figure 4-3 Menu Screen developed with jQuery Mobile deployed in iOS

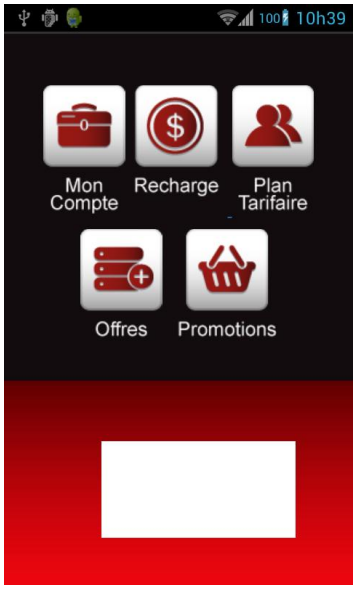

Figure 4-4 Menu Screen developed with jQuery Mobile deployed in Android

4.1.5 Account

The account screen displays the information about the user activities and events. To present this information this screen has two list views, where one of them can be selected to access extra information. Above each list is a label that identifies what that list represents. This screen will have a black background and a top toolbar with the title of the page and three buttons: one to open a side menu, one to return to the page menu and other to refresh the current page. The aforementioned interface is represented in the Figures 4-5 and 4-6, deployed in iOS and Android respectively.

To replicate the top toolbar was used the "header" element of JQM, that is automatically placed at the top of the screen. Inside the header was put a to center the title that is displayed using a <h2> element. After the heading, was added an <a> element with the data-role = "button".

The JQM button widget offers the possibility to configure many options about the button design and functionality. In this case it was used the option data-icon to define the icon to be used on the button; data-iconpos to define the position of the icon in the button, but since the header bar of the button will not have any text the option selected was "notext"; data-theme to define the theme of the button, which in this case is the same used in the login button theme and in the header. One important thing to have in mind about the header is that it is possible to set the position of the button in it through the definition of the class attribute in the buttons [52]. However when two buttons need to be on the same side of the toolbar, simply using the JQM class will not work, since the two will overlap. A nice workaround to this problem is to define a with the class attribute to configure the side of the element in the toolbar and put the two (or more) buttons inside of it. The classes used to define the position of the button are specified in the JQM CSS file. The details explained above about the implementation of the header are displayed in the next code snippet.

```
<div data-role="header" data-theme="f" >
      <h2>Mon Compte</h2>
      <a href="#" id="btn_slider_acc"
            data-role="button" data-icon="bars"
            data-theme="f"    data-iconpos="notext"
            class="ui-btn-left">
      </a>
      <div data-type="horizontal"
            style="vertical-align: middle"
            class="ui-btn-right">
            <a href="#" data-role="button"
                  data-icon="refresh" data-theme="f"
                  data-iconpos="notext" data-inline="true"
                  id=”btn_refresh”>
            </a>
            <a href="#" data-role="button"
                  data-icon="home" data-theme="f"
                  data-iconpos="notext" data-inline="true"
                  id="btn_menu_acc">
            </a>
      </div>
</div>
```

The lists in the screen use a data-role list view that enables the developers to easily create mobile focused lists. The first list is a summary of the events made by the user and therefore has inside each list item (<li>) a link element to the details of the event (<a>), as seen in the code snippet bellow. The second list is a regular JQM list view with the information about the balances of the account. Both lists are filled during the loading of the screen, when the web services with the information about the account are accessed.

```
<ul data-role="listview" data-theme="c"
      data-inset="true" id="account_info">
      <li data-role="fieldcontain">
            <a href="#" id="calls_value">
                  <b>Appels</b>
            </a>
      </li>
      …
      <li>
            <a href="#" id="internet_value">
                  <b>Internet</b>
            </a>
      </li>
</ul>
```

Once again it’s possible to notice that none of the anchor elements in the list refer to a page in the “href” attribute, because the tap events are handled by a JavaScript listener.

As stated before, to fill both lists it's necessary to access the Self-Care web services to retrieve the information. To give the user a better experience the ideal is to have the lists already filled when the page is shown to the user and to accomplish this with jQuery it is possible to access the web service before the page is displayed to the user. This can be done by configuring the event "pagebeforeshow" like in the next code snippet.

```
$('#account_page').on("pagebeforeshow", function(event){…}
```

Both HTTP REST request are done in this function and the data (in JSON format) is retrieved from the response. The first request done gets the list of events the user performed and with the response from the server, the events are summarized with the number of calls and their duration, the number of SMS and MMS and the amount of data consumed with Internet on the phone. This summary of events then is added to the list. To see the details about each type of event, the user can tap on the respective list item and a list of each event of that type is displayed.

After this, the second HTTP REST request is done to retrieve the balances from the services of the account. As in previous request to the server, the response is in JSON format and from there is retrieved the name and value of each balance, which are then added to the list.

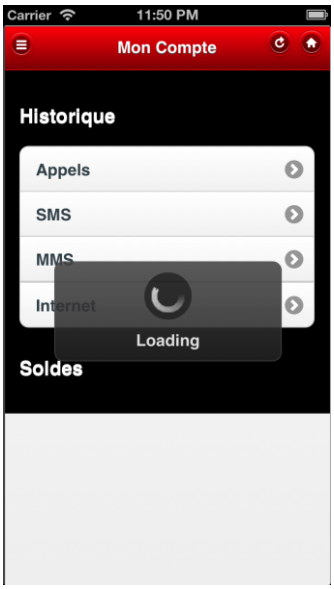

Figure 4-5 Account Screen developed with jQuery Mobile deployed in iOS (loading data)

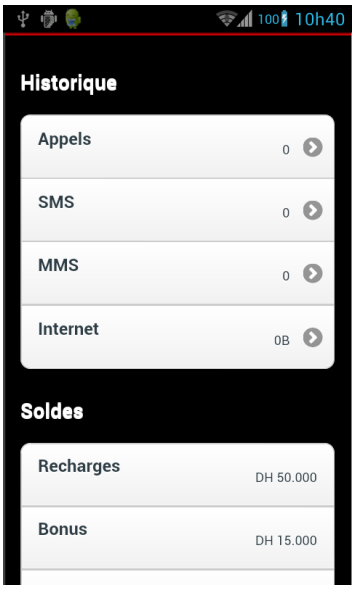

Figure 4-6 Menu Screen developed with jQuery Mobile deployed in Android (Data already loaded)

4.1.6 Recharge

The recharge page is the simplest page of the entire application. This page is composed by a header toolbar, a textbox and a button. Every UI component is similar to the ones already used in previous screens. In regard to the interface the only thing to notice is that the header toolbar only has the sidebar button and the home button as seen in the Figures 4-7 and 4-8.

To use the recharge function the user needs to put the code in the textbox and press the “Recharge” Button. When this is done, the mobile application will handle the “vclick” event on the button and make a HTTP REST request (POST method) to the server, which will have the user ID and the code as parameters. The request is done using the same jQuery function mentioned earlier ($.ajax) but with the type set to “POST” instead of “GET”. While the request is being processed a loading screen is shown to the user. If the recharge is successful a positive message is received, otherwise the server returns an error message. In each of the two situations the user will be notified about the result of the recharge operation through a popup window, opened using the code exemplified bellow.

```
$('#popup').popup("open");
```

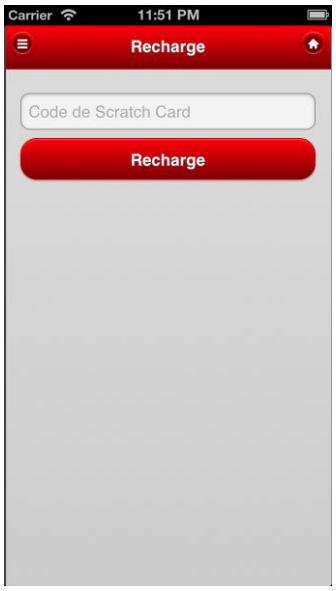

Figure 4-7 Recharge Screen Developed with jQuery Mobile deployed in iOS

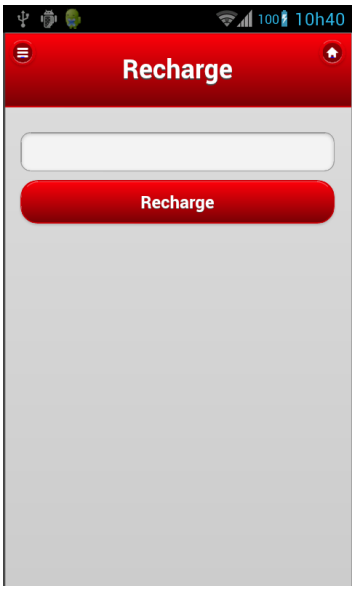

Figure 4-8 Recharge Screen Developed with jQuery Mobile depoyed in Android

4.1.7 Profile

The profile page gives the user the possibility to see all the mobile plans he can choose to his phone. To display this information is used a list that will display a checkmark on the list item that represents the plan used by the user at the moment. All the list items can be selected, giving the user access to detailed information about the desired plan. Also, like in previous screens, in the top of the screen is a header toolbar with the title of the page and two buttons (side bar button and home button). In Figure 4-9 it's possible to see the representation of the interface described when the web service doesn't return any profile (deployed in iOS) and in Figure 4-10 the same interface but when the request to the server is successful (deployed in Android).

To retrieve the information necessary to fill the list, it's necessary to send two HTTP REST requests (GET method) to the server, which are made before the page is shown on the screen. To give some feedback to the user about the information being retrieved, a loading mask is shown on the screen until all the data is available in the list. The requests will fetch the user profile information (which contains the plan the user is using) and the list of available mobile phone plans. Like in all the previously mentioned REST requests discussed, it was used the jQuery function $.ajax to send the requests to server. In the case of the request for the user information, it is necessary to add the User ID to the request, so the server knows to which user the requests refers to. After the first request is sent and the server response arrives, the second request is deployed, regardless of the first request being successful or not. To ensure that the second request is always sent, it is used the attribute "complete" from the AJAX request. This attribute let the developer define a function to be performed after the AJAX request is finished and like said above, it is executed regardless of the result of the request. If the requests are successful, the information the server returns (in JSON format) is retrieved, parsed and added to the list. In the code snippet below it is demonstrated how to add an item to the list view.

```
var profile = item.Name;
lista.append($("<li/>",
{
      "data-icon" : "false"
}).append($("<a/>",
{
      'href' : "#",
      "text" : profile
})));
```

When one of the list items is tapped by the user, the application will transition to a details page. This page will display a small description about the phone plan and a button that let the user change to it (in case that it is not the plan the user already has). To make this button visible or not its "css" property is accessed in the JavaScript file and the value of the "visible" field is changed accordingly.

If the user wants to change the plan, after pressing the button, the smartphone application will make a HTTP REST request (POST) to the server with the user ID and the plan ID in the URL. While the application wait the server response, a loading mask is displayed on the screen and when the response is received the user will be informed of the result, either if it was successful or not, through a popup window which redirects the user to the profile view.

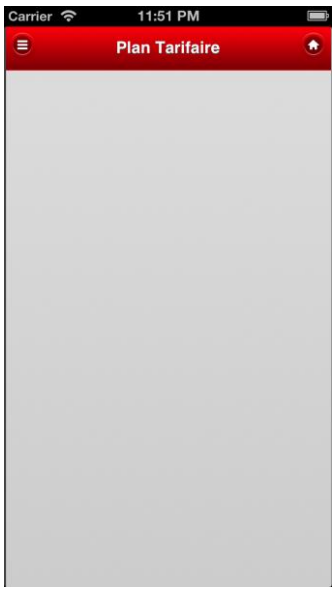

Figure 4-9 Profile Screen Developed with jQuery Mobile deployed in iOS (Profiles unnavailable)

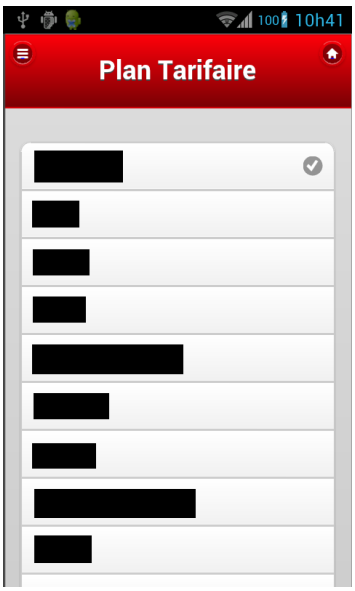

Figure 4-10 Profile Screen Developed with jQuery Mobile deployed in Android (Profiles available)

4.1.8 Offers

The offers page display to the user what service the operator has. Since this page shows the subscribed and unsubscribed services, there will be a list on the screen that is controlled by field set with two radio buttons. The way the page works is when the "Subscribed" radio button is selected the list of the subscribed services is displayed; when the user change the radio button to "Unsubscribed" the list on the screen will automatically change with an animation and display the unsubscribed services. In the Figure 4-11 is displayed the Offer page while still loading the offers and in Figure 4-12 is represented the same interface but with the offers already loaded. The code bellow represents how the field set with radio buttons was created.

```
<div>
      <fieldset data-role="controlgroup"
      data-type="horizontal" id="radio_op">
            <input type="radio" name="radio-choice-2"
            id="radio_sub" value="sub" checked="checked" />
            <label for="radio_sub">Subscribed</label>
            <input type="radio" name="radio-choice-2"
            id="radio_avai" value="ava" />
            <label for="radio_avai">Available</label>
      </fieldset>
</div>
```

Although the two lists are never displayed at the same time, the user is able to access any of them at any given time. To achieve this, the data to be displayed to the user is fetched with two requests to the server. The first HTTP REST request (GET) get the subscribed services and after the server response the information is parsed, saved and added to the list. While the information is displayed the second HTTP REST request (also a GET) is issued to the server to get the unsubscribed services. This way the user will have the illusion that all the data is already available to him, while in reality it is being fetched. As in previous pages the first HTTP request is made before the page is shown to the user and while the mobile applications is waiting for the server response, a loading mask is displayed to the user that gives him feedback about the request being made.

The list with the services can be tapped by the user to access details about it. The details page also has a button that let the user to subscribe the service if it is not subscribed already. This process is very similar to what the "Profile" view does, since subscribing to a

service is also made through an HTTP REST request (also POST) and the application will wait for the answer from the server to inform the user about the success of the operation.

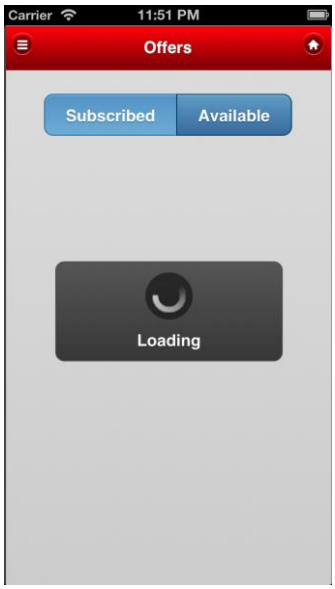

Figure 4-11 Offers Screen developed with jQuery Mobile deployed in iOS (Loading Offers)

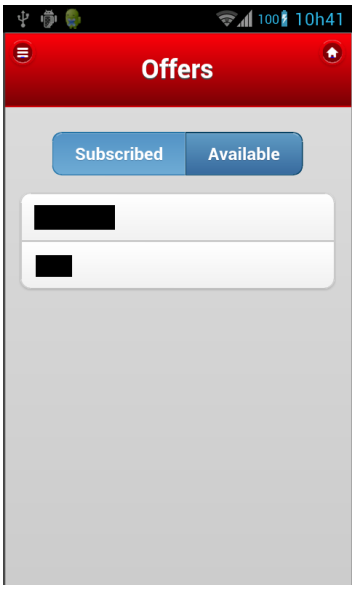

Figure 4-12 Offers Screen developed with jQuery Mobile deployed in Android (Offers Loaded)

4.1.9 Sidebar Menu

One feature that all the pages, besides the "Login" and the "Menu", have is the possibility to open a Sidebar Menu. This menu will give access to all the other pages of the screen without having to go back to the "Menu" page. The resulting interface is iOS and Android is displayed in the Figure 4-12 and 4-13.

To create this panel a jQuery Mobile widget "panel" is used. This widget gives the possibility to the developer to create a panel that will be hidden to the user and that can be activated through a set of function that JQM offers. The panel element has to be on the same level (sibling) of the header, body and footer of the page. Inside the panel the developers can set any content they wish like in a normal page. In this case, inside the panel is a list view, with 5 items, one for each of the options of the menu, as exemplified in the next code piece.

```
<div data-role="panel" id="sliderPanel_offers"
data-theme="a" data-swipe-close="false" data-dismissible="false">
      <ul data-role="listview" data-theme="a">
            <li>
                  <a href="#account_page">
                        <img  src="account.png"
                        class="ui-li-icon">Mon  Compte
                  </a>
            </li>
            <li>
                  <a href="#recharge_page">
                        <img src="recharge.png"
                        class="ui-li-icon">Recharge
                  </a>
            </li>
            …
      </ul>
</div>
```

The access to this menu can be made through a button in the header bar or making a swipe move on screen from left to right.

Like any other button in this application, when the tap event is triggered, a function in a JavaScript file catches the event and simply toggles the panel. Toggling in this case is a better solution instead of opening the panel, because this way the open and close events are handled in the same function, avoiding the need to handle each one of them separately.

Handle a swipe event is very easy using the JQM framework, since it gives the possibility to add to the HTML file listeners for the event “swiperight”. When this event is detected by the application, it toggles the panel, exactly as pressing the button.

Since almost every page on the application has this panel component it’s necessary to add the listener to the swipe event in each page.

```
<script>
      $(document).on("pageinit", "#home", function() {
            $(document).on("swiperight", "#account_page",
                  function(e) {
                        $("#sliderPanel_account").panel("open");
                  });
            …
            $(document).on("swiperight", "#promotions_page",
                  function(e) {
                        $("#sliderPanel_promotions").panel("open");
                  });
      });
</script>
```

In this case, as seen in the code snippet above, the function used to open the panel is the “open” function, since the swipe move only should open the panel and not close it, contrary to what happens with the side menu button that should toggle the menu.

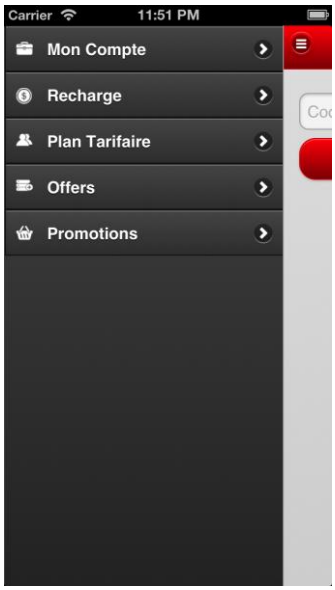

Figure 4-13 Sidebar Menu Developed with jQuery Mobile deployed in iOS

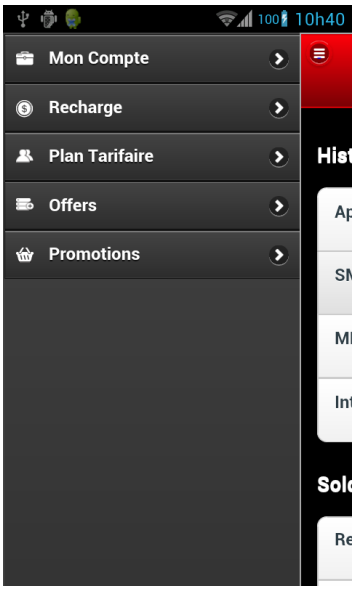

Figure 4-14 Sidebar Menu Developed with jQuery Mobile deployed in Android

4.2 Sencha Touch

4.2.1 Setup

The setup of a Sencha Touch project can be a lot more complicated than the setup of a jQuery Mobile project, but Sencha offers the developers the tools that try to make this process as simple and fast as possible. The most important tool is the Sencha Cmd, which for a proper development is absolutely necessary, since it can create, compile, package and deploy the application, with the desired configurations.

It is possible to work with Sencha Touch without using the Sencha Cmd tool, although not the ideal approach. To do this, developers must create themselves the MVC project structure and then include all Sencha Touch SDK in the project. This is not the best approach to the development process because with Sencha Cmd, the creation of the MVC structure, the inclusion of the ST SDK and the packaging are done in a much more efficient way. For example, when the Sencha Cmd includes the SDK in the project in the packaging process, it only includes the absolute necessary files to the project, making the project much smaller, something that doing "manually" by the developer can be a lot more complicated.

With all this in mind, using the right tools is a great help when working with Sencha Touch, since the setup of the project resumes to its creation through the Sencha Cmd command:

```
Sencha generate app AppName /path/to/AppName
```

As said in the introduction to Sencha Touch, this command have to be run inside the Sencha Touch SDK folder, otherwise, before the "generate" it's necessary to add "-sdk /path/to/Sencha-touch-sdk" [44].

Unlike JQM, where all JavaScript and CSS files were included in the main HTML file, in ST the developers must put all the information about external resources they want to use in the app.json file. This file will include the path to external JavaScript files, CSS files and other resources like images, icons, data, etc.

4.2.2 Login

To create the Login page, its View must be created in the first place. As stated in the Sencha Touch introductory chapter (chapter 2.4.2), in this file are configured all the UI components of the page.

There are three very important settings that need to be defined when starting a View: the “extend” field, which defines the parent class that the view extends; the “xtype” field, which defines the shortcut to the view name; and the “requires” field, which defines the UI component required by the view, so they are included when packaging the application. All the pages in this application will extend the component “Container” that, as the name suggests, is a component that can contain other components in it. To make it easy to refer each page, their “xtype” will be the same as its name, but beginning with lower case.

To create this page, it were required some component like: a label (title), a text field (username), a password field (password), a field set (will group up the username and the password fields), a button (login button) and a another container with layout “vbox” that helps to position components horizontally, in this case the field set and the button.

To add this components to the view, two methods are available: either add them to the container in the initialize field (this field let the developer define a function that will be triggered when the view is initialized); or inside the configuration field of the container where, in the “items” field, all the components of the UI can be declared. In this application both methods were used for testing purposes, but in this page, more specifically, the components were added in the initialize function as demonstrated in the code snippet bellow.

```
Ext.define("SelfCare.view.Login", {
      extend : "Ext.Container",
      …
      initialize : function() {

      this.callParent(arguments);
            var title = {
                  xtype : "label",
                  …
            }
            …
            this.add([title, box]);
      }
…
```

The log in functionality is handled in the Login controller file. The button component on the view, delegate the handling of the function to the controller. In the controller it's necessary to make reference to the view (using the xtype declared) and to the event that is being handled. Since the events are triggered in the views and handled in the controllers, the name of the events in the controller has to be the same that was used in the view, when calling the controller.

In the function that handles the button tap event, a loading mask is shown to the user using the Ext.Viewport.setMasked method. After this, the HTTP REST request is made (GET method) using the Sencha Ext.Ajax component. In this component it's possible to define the URL of the request and the success and failure callback functions. In case of success the current view changes to the menu view. In case of failure a message alert is displayed to the user to notify him that the log in process has failed. The process described above is exemplified in the next code extract.

```
onLoginCommand : function() {
Ext.Viewport.setMasked({
            xtype : 'loadmask',
            message : 'Loading'
      });

      var me = this;
      Ext.Ajax.request({
            url : "url do servidor/",
            success : function(response) {
                  Ext.Viewport.animateActiveItem(me.getMenu(),…
            },
            failure : function(response) {
                  Ext.Viewport.setMasked(false);
                  Ext.Msg.alert("SelfCare", "Login failed.",
Ext.emptyFn());
            }
      });
},
```

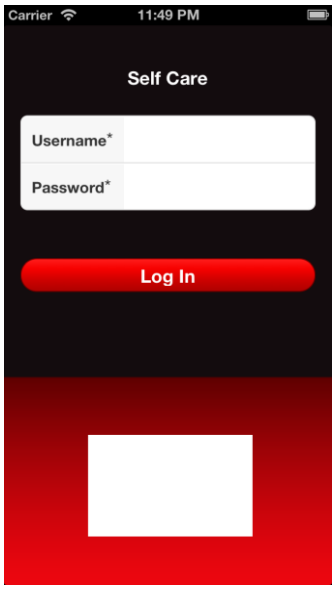

Figure 4-15 Login view Developed with Sencha Touch deployed in iOS

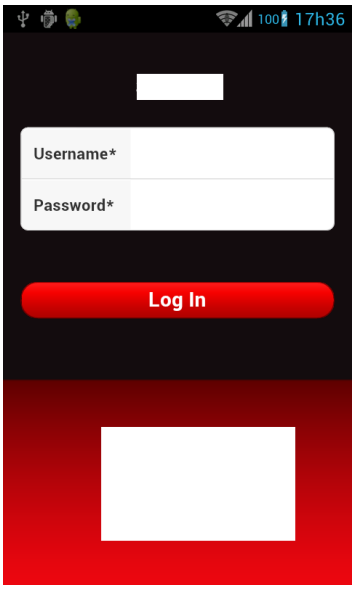

Figure 4-16 Login view Developed with Sencha Touch deployed in Android

As seen in the Figure 4-15 and 4-16, the button has a very similar colour of the button in the original application, however Sencha Touch default button are not coloured this way. To accomplish this, the SASS files generated by the Sencha Cmd were used. When Sencha Cmd creates an application, in the SASS folder, inside the resources folder, there is a empty SASS file which is intended to be used if the developers wants to change

the overall look of the Sencha Touch visual elements. To make the button look similar to the one in the original application it was necessary to change the base colour, the active colour and the base gradient of the scheme. Since the file generated change the whole theme and then is added to the project in a way that override the default theme (added after the default CSS file), there was no need to change anything when declaring the button, since the new theme is now the default theme for buttons and toolbars.

4.2.3 Menu

In the Menu view, it was necessary to have a button for each of the options and a particular thing is that each button, when pressed, should change its background to give the user a feedback that the button is being pressed.

In Sencha Touch it's possible to set the CSS of each element, making the procedure of customizing elements easier. This can be achieved through the configuration "cls". Also, the button component let the developer set a different CSS class for the button when it is being pressed, what makes the task of giving the user a feedback on the touch event much easier, has seen in the code snippet below.

```
var account = {
      xtype : "button",
      cls : "accountBtn",
      handler : this.onAccountTapped,
      pressedCls : "accountBtnPressed",
      scope : this
};
```

To position the buttons side by side, is used a container with layup type "hbox" and the "spacer" component between each button. The "spacer", as the name suggests, let the developers put some space between components. This is very useful to help to tweak the disposition of every UI elements. The interface of this page is represented in the Figures 4-17 and 4-18.

Every button has a listener which trigger an event in the controller. In the Menu controller, the only thing to do is the transition to the selected view. To use a transition is necessary to call the Ext.viewport component and the function animateActiveItem(). The

arguments to this function are the view to where the transition will be made and the animation made during the transition [53] as seen in the code snippet.

```
Ext.Viewport.animateActiveItem(this.getServices(), {
      type : 'slide',
      direction : "left",
});
```

Note that the animation can have much more configurations defined, like listeners for when the animation ends (it can be a great way to optimize transitions in the application) and much more, but the code above is just the simplest example because more complex details about this will be discussed when another view makes use of those features.

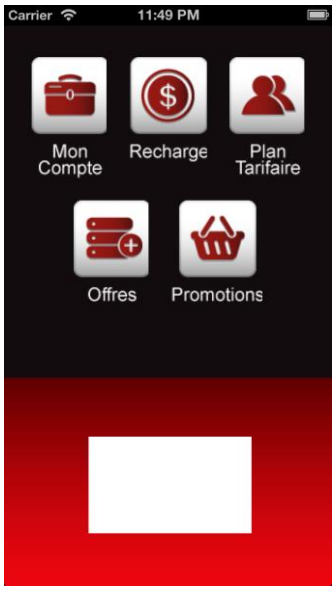

Figure 4-17 Menu view Developed with Sencha Touch deployed in iOS

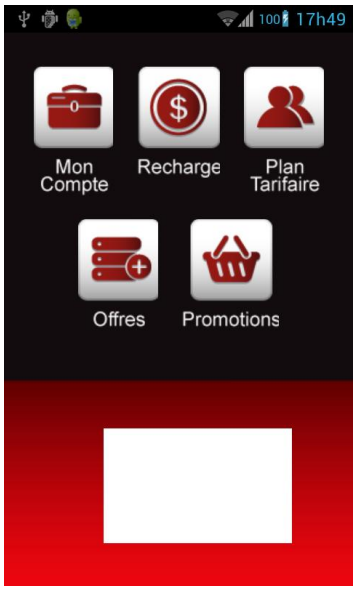

Figure 4-18 Login view Developed with Sencha Touch deployed in Android

4.2.4 Account

Unlike JQM, this view will have a button that will control what information is displayed to the user. In the original application and in the JQM application, the summary of the events performed by the user and the list of balances for each service were displayed in two different lists, which were on the screen at the same time. However in Sencha Touch, there is a segmented button controlling if the summary of events is being displayed,

or on the other hand the list of services balances is displayed. This approach was taken mainly because there were difficulties in recreating the original UI and therefore this option was chosen since this approach is already used in other screens of the application, making the overall look of the application consistent with each other.

Besides the segmented button, the view also adds a toolbar, attached to the top of the screen, with the title of the view and three buttons (one to open the side menu, one to return to the main menu and another to refresh the view). These buttons are declared inside the toolbar in conjunction with a spacer element, which was added to replicate the disposition of the button (one button on the left, two buttons on the right of the toolbar). The final result of this interface is displayed in the Figures 4-19 and 4-20.

The summary of the events performed by the user are displayed in four text fields elements (configured to be read only). The list of the services balances is displayed on a list element. The control of the elements displayed is done through the use of a container with a layout card, declared as seen in the code snipped below. This type of containers can have several items inside them, but only one will be displayed at each time, making it ideal in this situation, since when the segmented button it's activated, the container is changed accordingly.

```
{
      xtype : "container",
      layout : "card",
      itemId : "accountContainer",
      flex : 10,
      items : [{
            …
      }]
}
```

To fill both the summary of the events performed and the services balances, it was necessary to access to the supplied web services. The access to them was made using a HTTP REST request (GET method), which is sent while the page is being loaded. This first request will retrieve the list of the events performed (using the function Ext.Ajax), parse them, summarize them and display them on the text fields. The text fields have a listener to tap events, so the user can have access to a detailed list of all the events the user performed of each type. The second request is sent through the load function of the store, which saves the list of the services balances. All these actions are made by the controller of the Account view.

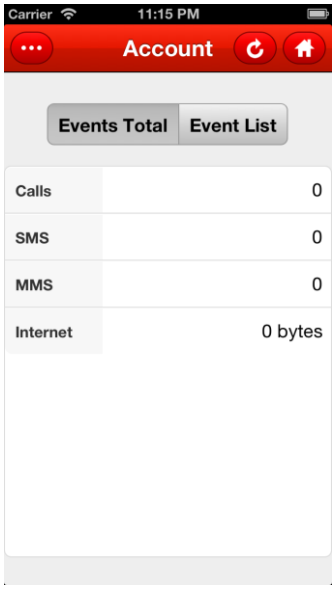

Figure 4-19 Account view Developed with Sencha Touch deployed in iOS (no events retrieved)

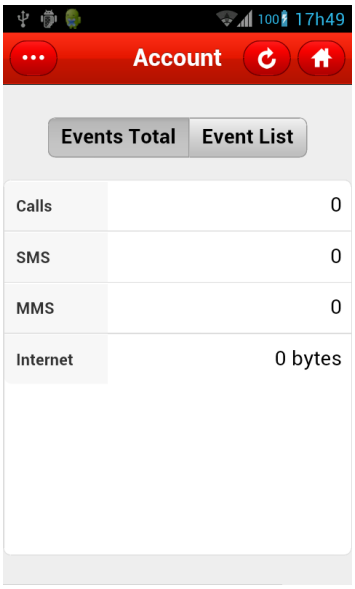

Figure 4-20 Account view Developed with Sencha Touch deployed in Android (no events retrieved)

4.2.5 Recharge

To replicate this view it was used a toolbar, docked to the top of the screen, which has two buttons and two containers with layout "hbox" (to center the components): one with a number field inside (where the user will put the code); one with a button.

To give the button the rounded corner look, Sencha Touch let the user define the "ui" configuration. This configuration defines some predefined styles for the button, like "forward", "back" or "round", making easy to have different type of button in the UI, as seen in the buttons of Figures 4-21 and 4-22.

The numerical field is used, instead of a normal text field, so the user only can put numbers in that textbox. This can be seen in the image bellow, where the keyboard shown to the user to input the code is numerical as seen in Figure 4-22.

The recharge button has a listener for the tap event. As any other event in this Sencha Touch application, it is delegated to the controller. In the controller a command to display a load mask to the user is used, to inform him that the recharge process is in progress. After that the Ext.Ajax class is used to make the HTTP REST request and the success and failure of the operation are handled by a function that can be defined in the configurations of the request (similar to the jQuery $.ajax). As in case of success or failure,

a pop-up notifying the user of the outcome of the request is displayed. The only difference between the handler of the two results (besides the message to the user) is that if the request is successful, when the user press the “Ok” in the pop-up it will redirect the user to the menu view, otherwise pressing the “Ok” button will only make the pop-up disappear and the user will still be in the “Recharge” view. The code necessary to implement the procedure explained above is displayed in the next code snippet.

```
…
success : function(response) {
Ext.Viewport.setMasked(false);
      Ext.Msg.alert("SelfCare", "Recharge Successful", me.goMain());
},
failure : function(response) {
      Ext.Viewport.setMasked(false);
      Ext.Msg.alert("SelfCare", "Recharge Failed.", Ext.emptyFn());
}
…
```

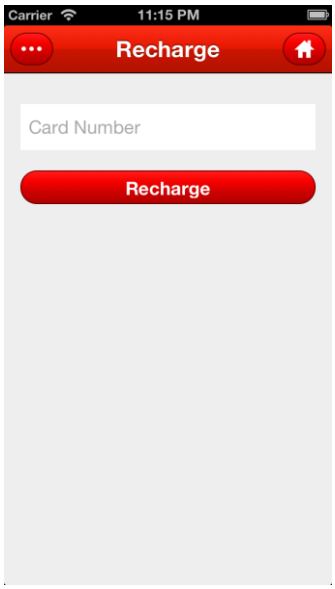

Figure 4-21 Recharge view Developed with Sencha Touch deployed in iOS

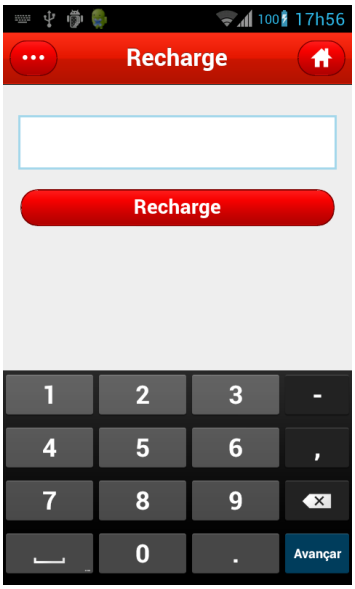

Figure 4-22 Recharge view Developed with Sencha Touch deployed in Android

4.2.6 Profile

The profile view, like all other views (beside the Login and the Menu) will have a toolbar with two buttons and the title of the page, docked to the top. This page will also have two lists and the reason for having two instead of one like in JQM, is that in Sencha Touch, the data of a list have to be supplied by a Store and since the list of phone plans available and the plan selected are fetched from different web services, two stores had to be created. This problem will make the view not look exactly like the original one, but the all the information necessary to the user will still be present on the screen, as seen in Figure 4-24. In Figure 4-23 it is possible to see that the interface of the application, when no profiles were retrieved, is very similar to the interface developed using JQM.

Both of the list show the name of the plan and all the items are clickable, to give access to details about it (same behaviour as in the JQM application). The plan details view is a very simple, with a label with the name of the plan, an html element with it description and a button to change it (if that's the case).

In terms of functionality, the application loads the data to the stores when the view is activated (ST gives the possibility to listen to this event). Since the URL of the web service, the format of the data and the type of request is already defined in the data Model, to get the data it only required to use the load() method of the Store associated to the Model. Then to format the content of the list, the "itemTpl" (which stands for item template) configuration enables the developer to set what data is in each list item and how it's formatted (this can be done using HTML in the configuration field).

One great advantage about saving the retrieved data in a Store is that the information is already available, when the user wants to access it. This happens mainly because the listener on the tap event on a list item returns the record that was pressed, so beyond the plan name, all the information saved in the Store about the plan is available as demonstrated in the code snippet below.

```
onItemTapped : function(list, record, target, index, evt, options) {
            this.fireEvent('profileTapped', this, record);
},
```

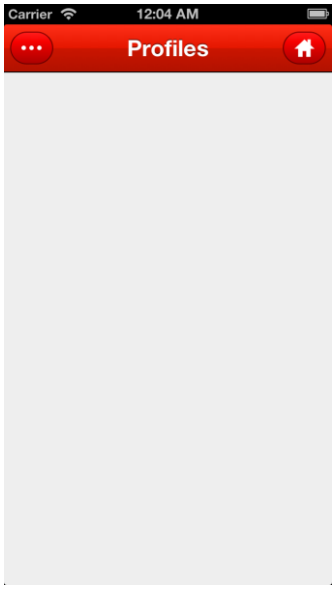

Figure 4-23 Profile view Developed with Sencha Touch deployed in iOS (Profiles unnavailable)

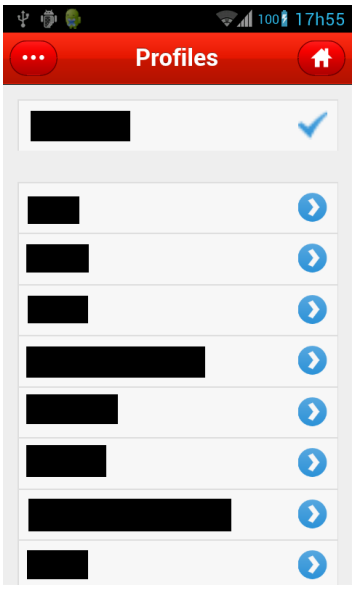

Figure 4-24 Profile view Developed with Sencha Touch deployed in Android (Profiles available)

4.2.7 Services

The services view, like the profiles view, will have, besides the view and controller files, a store and model files, since it will need to access the web services that return the available services to the clients and then store them. But unlike the Profile page, the Service list will only have one list that will change its store every time the user presses one of the buttons that control the view.

The interface will be composed of a toolbar with two buttons docked to the top, a segmented button with layout “hbox” that will control the content of the list and the list with the information about the available and subscribed services. The toolbar is exactly the same as in the previous screens. The segmented button element enables the developer to group up buttons, which in this case is very useful to create a similar look to what the original application have as seen in Figures 4-25 and 4-26. Declaring a segmented button can be done using the code below.

```
{
	xtype : "segmentedbutton",
	itemId : "segBtn",
	layout : {
		type : "hbox",
		pack : "center"
	},
	items : [
	{
		text : "Subscribed",
		itemId : "SegmSub",
		pressed : true
	}, {
		text : "Unsubscribed",
		itemId : "SegmUnsub"
	}]
}
```

To control what content is displayed on the list, the segmented button will have a listener to the toggle event that will change the source of the list data, which is defined in the "store" configuration field. Although the two stores fetch their data from different web services, since the information retrieved from each request is similar, it's possible to change the store associated to the list without changing it formatting, with a simple if condition as seen in the next code snippet.

```
onSegToogledCommand : function(btn) {
	if (btn.config.text == "Subscribed") {
		var myStore1 = Ext.getStore("SubServices");
		this.getServicesList().setStore(myStore1);
	} else {
		var myStore2 = Ext.getStore("AvServices");
		this.getServicesList().setStore(myStore2);
	}
}
```

To retrieve the data to fill the lists, the process is similar to the one used in previous screens, which means the data is fetched while the view is initialized, using the store load() function. One important thing to notice is that the first store loaded is the one that has the data of the subscribed services because this is the list that is displayed when the page is first presented to the user. After this store load is successful, the load of the unsubscribed services begins. This is done, like in the JQM application, so the user has some of the data available, without having to wait a lot of time for data that he does not need at that moment, making him believes the application is faster.

Like in the Profile page, all the list items have a disclosure button, that let the user access detailed information about the service, and if the service is not already subscribed it is possible to do so. This detailed information is displayed in a new View that has a panel that contains the details about the service and a button that let the user subscribe to that service (if is not already).

To subscribe a service it's necessary to make a HTTP REST request (POST method) to the server and while that process is on-going, a loading mask is displayed to the user. When the server answer arrives, the user is notified through a pop up message about the result of the request.

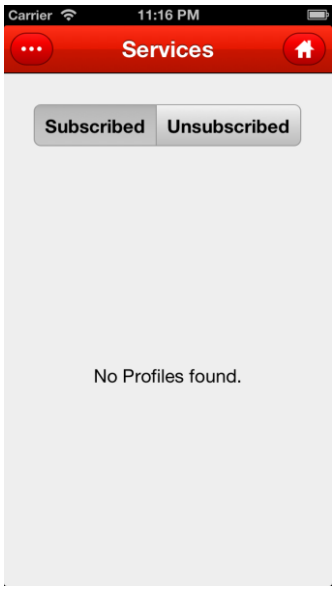

Figure 4-25 Services view Developed with Sencha Touch deployed in iOS (Services unnavailable)

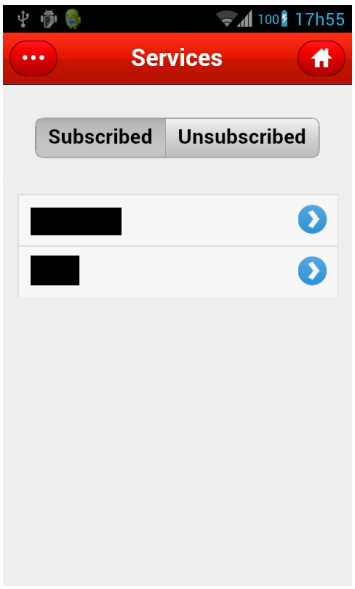

Figure 4-26 Services view Developed with Sencha Touch deployed in Android (Services unnavailable)

4.2.8 Sidebar Menu

As in the original application, in the ST version there is also a sidebar menu in all the views, except in the login and the menu. Contrary to the JQM, in Sencha Touch there isn't a mechanism that directly emulates the behaviour of the native application. However a work around to this problem is possible to achieve with the tools that Sencha Touch offers, having a good final result, very close to the native application, as seen in Figures 4-27 and 4-28.

Since the ST doesn't have a similar UI component to a side menu or side screen, to create the side bar it was used a normal container with a list containing each of the menu

options. Since in ST all the lists must have a store associated, one was created with static data with the name of the option on the menu and the respective path to the icon that represents it. This container is added to the view like any other component, but will have a smaller z-index value like the main container of the view. This will make the sidebar menu hidden at first.

The sidebar menu can be activated by pressing the button on the toolbar or by making a swipe movement with the finger on the screen from left to right.

When pressing the button the application will verify if the side menu is already open and if it is the main container will be moved to the beginning of the screen (offset 0), if not the side menu will "push" the main container to the right. This "pushing" is made using the property "draggable" of the main container. This property drags the main container to the desired offset, showing the side menu that in fact is below the main container.

As for the swipe move, what will in fact do is drag the main container to the right, displaying the side menu that is below it. One nice detail about this is that if the user already dragged the main container more than 50% of the screen width and stop the dragging, the main container will be dragged the rest of the way. This calculation of the drag direction is made by listening to the "dragbegin" event and the "dragend" event on the main container.

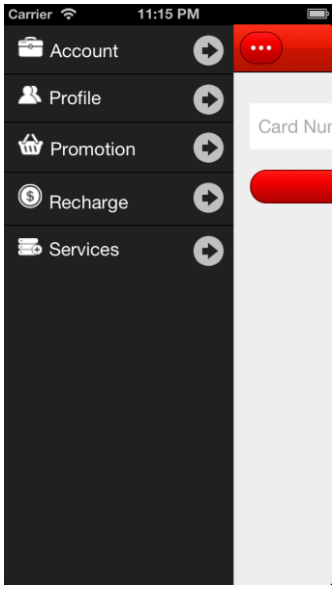

Figure 4-27 Sidebar Menu Developed with Sencha Touch deployed in iOS

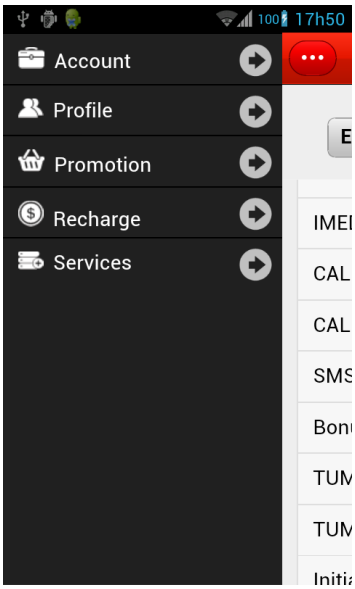

Figure 4-28 Sidebar Menu Developed with Sencha Touch deployed in Android

4.3 Windows Phone

Although the main test platforms of this dissertation were Android and iOS and although SelfCare don't have a Windows Phone version, it was also interesting to see how the code already developed would behave in this platform.

Since this was not original intended, any problem with the implementation were not fixed, since the problems themselves were something very interesting to analyse.

Even before the tests were made, the fact only that Windows Phone 8 (WP8) uses a different rendering engine than Android and iOS could be a source for problems. However the Sencha Touch had almost the same look in WP8 as it had in Android and iOS as seen in Figures 4-29 and 4-30. As for the jQuery Mobile version, the interface loses some of its configuration, something very noticeable in Figures 4-31 and 4-32. This indicates that the code would need to be reviewed in order to be possible to use the application.

Apart from interface issues, the functionalities seemed to work as intended. This is a great result, since these types of applications are mainly focused in being deployed in the greatest number of platforms possible (if profitable).

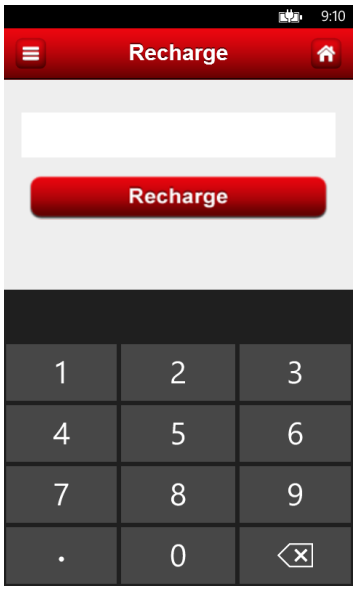

Figure 4-29 Recharge view Developed with Sencha Touch in Windows Phone 8

Figure 4-30 Side Menu Developed with Sencha Touch in Windows Phone 8 (bugged)

Figure 4-31 Recharge view Developed with jQuery Mobile deployed in Windows Phone 8 (formatting bugged)

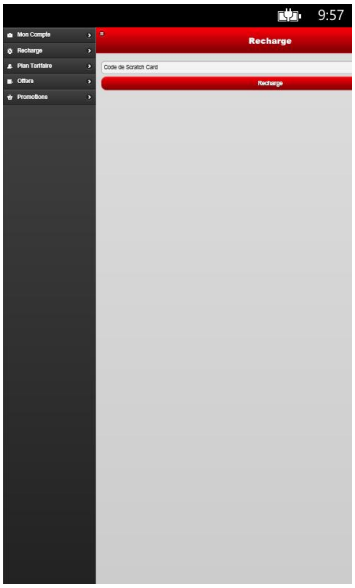

Figure 4-32 Side Menu Developed with jQuery Mobile deployed in Windows Phone 8 (formatting bugged)

4.4 Conclusions

At the end of the development of this application, it is very important to draw conclusions about how each framework responded to the features that were necessary to develop. Furthermore, it is also important to analyse how much support and documentation each of the frameworks has available, either by the community or by the framework developer itself, because this kind of support is very important, especially to new developers trying to start developing with these frameworks.

These conclusions will be divided in 3 segments, the first discuss the support the community gives to each framework, the second addresses the conclusions gathered while developing the application, evaluating more specifically the key features developed and finally the third segment refers and analyses performance indicators gathered from each of the applications developed.

4.4.1 Community Support

		jQuery Mobile	Sencha Touch
Repositories		1 882	976
Issues	**Open**	1 187	140
	Closed	5 161	291
	Total	6 348	431
Official Releases		33	Private Repository

Table 4-1 GitHub Statistics concerning usage of the frameworks

	jQuery Mobile	Sencha Touch
Stable	8	7
Total	25	26

Table 4-2 GitHub Statistics concerning usage of the frameworks

	jQuery Mobile	Sencha Touch
Active Questions	14 030	3 300
Unanswered	4 880	1 026
Followers	1 885	562

Table 4-3 StackOverflow statistics

	jQuery Mobile	Sencha Touch
Search Results	899	365
Quotes	124	22
Patents	6	3

Table 4-4 Google Scholar statistics

	jQuery Mobile	Sencha Touch
Threads	11 933	17 179
Posts	26 389	73 548

Table 4-5 Developer Forum statistics

	jQuery Mobile	Sencha Touch*
Twitter	13 700	19 640
Facebook	4 068	15 958
LinkedIn	1 583	4 589
Google+	588	1 550

Table 4-6 Social Network Followers

***Data gathered from the main Sencha social networks account, not exclusively related to Sencha Touch

	jQuery Mobile	Sencha Touch
No. of YouTube Videos	Approx. 36 500	Approx. 3 690
No. Google Search Results	Approx. 2 400 000	Approx. 869 000

Table 4-7 Google and YouTube hits

From the data gathered is easy to see that jQuery Mobile has a much greater support and visibility by the development community, this being most noticeable when take into account that in Stack Overflow, JQM has more than 3 times the questions of Sencha Touch, as seen in Table 4-3. However Sencha Touch own forum is used more frequently, which indicates that the developer can get valuable help from moderators, which are experienced ST users (Table 4-5).

Both frameworks have recent stable versions (Sencha Touch version 2.2.1 was released on 15th of April, 2013 and jQuery Mobile 1.3.2 was released on 19th of July, 2013) and are at the moment developing new ones. While jQuery Mobile let developers, who are interested, help in the development of the framework, Sencha Touch keeps the development private (relatively speaking to their GitHub projects). Since Sencha Touch keeps their project private it's not possible to compare information about the current number of issues, contributors, commits, etc., present in Table 4-1.

It's also important to notice that JQM is referenced in a much greater number than ST, even in the current year where JQM was referenced 264 times to the 87 of ST, according to Google Scholar (Table 4-4). Like many of the previous indicators this is a sign that more developers use JQM than ST, probably because jQuery is already used by many web developers, therefore making learn JQM very easy.

Finally it's curious to see that despite jQuery Mobile being more used by developers Sencha Touch has more followers in the social networks (Table 4-6). However JQM presents much more results in both YouTube and Google searches (Table 4-7).

4.4.2 Features Implementation

This section of the conclusion, about the frameworks, focus on the experience gained by the developer while developing this application. To display what was learned during the development, the application was divided in features and key elements. Despite

this application not being very complex, it has some features that can be considered important in every smartphone application and therefore important to implement.

The evaluation made of the software implemented was divided in two categories: the first one is an evaluation of the features implemented using some criteria suggested/required by PT Inovação and that were deemed suitable to evaluate the applications; the second was an overall evaluation of the application using criteria described by the Software Sustainability Institute in an article that illustrate how those criteria can be used to classifying software [54]. Taking a closer look to both criteria it's easy to see the ones used to grade the features are more straightforward and practical, something which was of interest of PT Inovação.

In Table 4-8 are exposed the features implemented and their classification. While analysing this, three aspects have to be taken into account: first the approach used was to implement the features in a way that the application could achieve the best performance possible; the second thing to have in mind is that the development of this application was done without almost any prior JavaScript, CSS or HTML5 experience by the developer; lastly it was intended to replicate as closely as possible the interface/functionality of the native application during the development process. Looking at the table it is possible to realize that some of the evaluation criteria, sometimes cannot be applied to some features, therefore the one that can be applied have a greater weight in the evaluation.

The table 4-8 is especially useful to someone who wants to develop an application, which mainly relies on a good UI and some basic web service interaction.

Table 4-9, as stated earlier, has some general properties about each frameworks and therefore gives a better overall look about what are the main advantages and disadvantages about each one of them.

To grade each table it was chosen a three grade system because in this type of study it was found easier to the developer evaluating the applications to have fewer grades available in order to obtain a more focused grading. So the grades obtained in each table for each item, reflect the personal experience and perspective of the developer, who was responsible by both applications development.

Feature	Evaluation Criteria	jQuery Mobile			Sencha Touch		
		Implementation Difficulty	Performance	Design / Look	Implementation Difficulty	Performance	Design / Look
List Scrolling		N/A	3	2	N/A	2	2
List Filling		3	3	N/A	1	2	N/A
Page Transition		3	2	3	3	2	3
Data Storage		3	2	2	2	3	3
REST Requests		3	3	N/A	3	3	N/A
Swipe / Drag Events		3	2	1	3	2	2
Side Menu Panel		3	1	1	1	2	3
Tap Events		3	2	3	3	2	3
Themes Use		2	2	2	1	3	3

Table 4-8 Comparison of features implemented using Sencha Touch and jQuery Mobile, according to three evaluation criteria (1- Poor, 3 – Good, N/A – not applicable)

		jQuery Mobile	Sencha Touch
Usability	**Understandability**	3	2
	Documentation	3	2
	Buildability	1	3
	Installability	3	2
Sustainability and maintainability	**Learnability**	3	2
	Community	3	2
	Accessibility	3	2
	Testability	2	3
	Portability	3	3
	Supportability	3	2
	Evolvability	3	3
	Interoperability	3	2

Table 4-9 General properties regarding jQuery Mobile and Sencha Touch (1- Poor, 3 – Good)

4.4.3 Performance Indicators

Finally, to get a more concrete evaluation about the applications performance, the memory usage and storage space of each of the application were measured, which can be seen has indirect performance indicators, assuming that the larger the memory requirements of an application, more slowly it performs. The objective of this measurement is to present some results that are not subject to the interpretation of the developers or the users, since the number presented below were retrieved from the same tests devices (same smartphones) and therefore can be used to gather some conclusions about the performance of the applications.

	Memory	**Disc Space**
Sencha – iOS	91,05 MB	11,9 MB
jQuery Mobile – iOS	62,15 MB	11,2 MB
iOS Native	36,52 MB	2,2 MB
Sencha – Android	76 MB	2,11 MB
jQuery Mobile - Android	72 MB	1,77 MB
Android Native	46 MB	1,52 MB

Table 4-10 Performance measurements made with SelfCare HTML5 and Native application

In the Table 4-10 above is possible to see that both the iOS and the Android native applications require less memory and less storage. If look exclusively at the iOS applications we can conclude that the native application is much smaller (5x times smaller) then the hybrid applications. This can be attributed to bad configuration in the iOS project, something that with some improvements could make the gap in storage necessary smaller. As for the memory used by the applications, the iOS native also requires fewer resources, using less than half the memory of the ST application. Looking only to the two hybrid applications, jQuery Mobile "wins" in both categories and in the memory usage is where the difference is more noticeable.

As for the Android application, the difference between the native application and the hybrid ones is smaller. Focusing only on the hybrid application, the gap in performance here also decreased, but once again the jQuery Mobile application presented better results.

What can be concluded from these values is that the Native application presented a better performance in both platforms (something that was expected), being this more noticeable in the iOS environment. As for the hybrid application, jQuery Mobile has an

edge in these performance indicators and although the difference is small, it can be decisive when choosing a framework to use, when developing with these technologies where the performance is a crucial point (more than usual).

5. Performance enhancement practices

When developing any software, performance is always a concern in the mind of the development team, therefore knowing some practices/tricks that can help make a program faster, smaller or more appealing are very useful.

In this chapter some of the practices learned while developing the first mobile application are explained. These techniques were implemented so the hybrid application developed had a similar performance to a native application.

5.1 General practices

While some techniques can be specific to each of the frameworks addressed, others can be applied to every hybrid mobile application. These are practices that should always be on the mind of a developer while programming a hybrid mobile application.

One of the most important issues when developing these kinds of applications is its size. One way to reduce the space of an application is to minimize the CSS and JS files. This simply consists in removing unnecessary characters, like white-spaces, making the files load faster. This process can be very easily done resorting to some online tools. However most of these sites only compress one file at a time, which makes this process very long and tedious.

When using JavaScript to access or change anything displayed on the screen it's necessary to find the DOM elements and then manipulate it accordingly. These type of searches, most of times, will not be a burden, performance wise, in your application, but if they are made many times, they will start to be noticeable in its flow. One way to avoid this problem is to, when knowing that some elements will be necessary more than a few times, save it in a variable, therefore avoiding to search the DOM.

Mobile application developers that chose to make hybrid application should have in mind the variety of mobile devices that can use their applications and the different hardware capabilities from each one of them. This is important because some features offered by HTML5 and CSS3 are more demanding to the smartphone and sometimes don't

give many advantages in user experience. Some of the features that developers might consider avoiding while developing hybrid applications are:

- 3D animations: transitions with this kind of animation can be slower on older, less powerful devices;
- Box Shadows and Gradients: they are drawn dynamically therefore actions like scrolling a list with lots of these features can feel sluggish.

When the applications being developed have to make CPU-heavy operations, their usability might decrease, since the interface will become non-responsive. To avoid this problem, HTML5 offers the developers the possibility to use web-workers. Web-workers are JavaScript code that can run simultaneously with other JavaScript files, avoiding the interface becoming unresponsive while being executed. However, developers which use this tool have to keep in mind that web-workers cannot modify any interface related feature directly (they cannot access the DOM), they can only return the data processed to the main JavaScript and then that data can be handled accordingly and displayed to the user.

The visual aspect of mobile applications is very important, because this characteristic can be instrumental in attracting users to the product. The way the visual elements of an application interact with each other (especially when they are complex) is very important to give the user a good experience. To enhance the way these interactions, transitions and other visual operation look, developers can enable the hardware-acceleration from the mobile browser. This "trick" will force the browser to use the GPU in any process that draw on the screen. Enabling this features can be done in some configurations, specific to each platforms targeted, but an easier way to accomplish this, is by tricking the browser into enabling itself, using the CSS code presented in the snippet bellow, which triggers this feature:

```
-webkit-transform: translateZ(0);
```

Note: the code above target only browser with webkit as their rendering engine.

Another small trick that can make increase the usability of the application is putting the CSS in the header of the HTML file. This will give the user the illusion that the page is loading faster, since the page will load progressively without the necessity to redraw elements if the style changes [55].

Many more techniques that are usually used in improving the performance of websites can be applied in this kind of mobile applications, since the technology used is the same, however it's important to look to them with criteria, since a hybrid mobile applications have different features of a website. Furthermore, analysing platform specific techniques, when developing an application that will later be put on the market, is also very important, however these methods were not addressed in this dissertation because they did not fit in the scope of this work.

5.2 jQuery Mobile Specific practices

When applying these techniques, knowing the limitations of the technology being used is very important. This is why it is important to have specific practices to specific technologies (in this case specific frameworks).

One issue while using jQuery Mobile is that touch events display a 300ms delay before they are handled by the framework and although this can be useful to avoid confusion between tap and swipe or drag events, this also make the interface reaction look a lot slower. One way to avoid this delay is by handling the mouse "down" event. This event triggers when the application detects the tap on the screen event before the user lift the finger. Handling this can be tricky, because it will also be necessary to handle the mouse up, but will make the delay very minimal on tap events. Alternatively, there are some JavaScript plugins that try to fix this issue, like FastClick.js or Hammer.js.

In jQuery Mobile when events are triggered, there are default actions performed by the framework, which sometimes might be undesirable by the developer and might bring the performance of the application down. To avoid this, jQuery Mobile offers a method, that is presented in the next code snippet, which prevents these actions from being triggered. Using this method gives the developer a better control over the application, because he can choose which actions are performed and "overwrite" the ones that are unwanted. However is necessary to be careful when using this function, since it can disable an action that is actually wanted by the developer.

```
event.preventDefault();
```

5.3 Sencha Touch Specific practices

One of the main advantages in using Sencha Touch is how easy to use some good practices that can improve performance. Using Sencha Cmd can make the process of minimizing files, import only necessary features of the framework and package the applications very easy, since it is all done automatically when the project is built.

When using Sencha Cmd the developers has to keep in mind that some files need to be configured in order to have a more efficient packaging process. The files that are important to configure are the app.json which contains the configurations for development, the packager.json which contains the configuration for native packaging and the sencha.cfg file which contains some specific Sencha Cmd options (most of the times not as important as the previous two mentioned files).

Lists in Sencha Touch, by default can be scrolled beyond their boundaries, making their performance slower than normal. Since many applications use lists, in one way or another, avoiding this behaviour is very important. To disable this feature, the developers need to configure the lists scrolling configuration. The following code will make the lists only scroll to where they have items, making them perform better.

```
scrollable : {
      direction : "vertical",
      directionLock : true,
      momentumEasing : {
            momentum : {
                  acceleration : 30,
                  friction : 0.5
            },
            bounce : {
                  acceleration : 0.0001,
                  springTension : 0.9999
            },
            minVelocity : 5
      },
      outOfBoundRestrictFactor : 0
},
```

6. Rama – advanced hybrid application

6.1 Introduction

After the development of the first application and some conclusion about the HTML5 frameworks were drawn, it was necessary to make a more complex application to have a better understanding of the capabilities of the chosen framework and from HTML 5 itself. To deepen the knowledge about these features, it was chosen an application named Rama. Rama is a tourism focused application that uses augmented reality as its differentiating factor.

Based on the conclusions previously drawn about jQuery Mobile and Sencha Touch, the later was chosen to develop this application. From the advantages mentioned above about this framework, the most important in the selection process were the fact that Sencha Touch uses an MVC pattern, that make the development more structured and organized and the availability of tools that help in the deployment process (Sencha Cmd). But contrary to the development of Self-Care, in this case the focus is not on what the framework has to offer to the developers, but in what way the capabilities of the framework can work together with some key features of HTML 5 itself. Accomplishing this could make the development of mobile application more viable, giving the user a smoother experience, while using standard web technologies.

As said before, Rama uses augmented reality to stand out among the possible competition it faces, but besides that another important features it offers is the integration with Google Maps (something extremely useful in a tourism application). Both these features will use Points of Interest (POI) as a way to guide the user through the city of Aveiro (the application is specific to this city) and show him what is relevant in terms of tourism. These points can be accessed through a web service which retrieves the name of the point, the type, the ID, an image associated with the point and its geographical coordinates.

6.2 Wikitude

Having an augmented reality view is what makes Rama stand out from among similar applications. Augmented reality is a very nice feature in applications that are focused in showing Points of Interest (POIs) to the users. Since augmented reality is the inclusion of media elements (images, text, videos, etc.) in a real-time view of the real world, it's necessary to have access to a smartphone camera.

Using a smartphone camera in a hybrid or web mobile application is not an easy task, because, at the moment, the current default smartphone browser doesn't support this feature. Consequently, it was necessary to use a plugin designed to implement augmented reality features in smartphone applications.

The plugin chosen to implement the augmented reality view for the Rama application was Wikitude. Despite Wikitude help the developer implementing augmented reality, in a hybrid mobile application it needs to be used in conjunction with Phonegap. Phonegap is an application container, which let developers deploy mobile applications created with web technologies. Using Phonegap instead of simply launching an hybrid application through a call to the mobile browser offer the possibility to easier interact with some smartphone features like media, network, compass, etc.[56]. On the other hand, this also means having another layer of code, which can make the application "heavier". In this case, what Wikitude does is take advantage of the calls that Phonegap make to the native resources of the smartphone (camera for example) and use them without having to worry about the implementation details.

However using Wikitude needs some setup, but it is simple and can be easily found in their developer website [57]. One downside to using Wikitude, at least at the moment, is the fact that if a developer intends to use it as a Phonegap plugin, he will only have the possibility to deploy the application in Android and iOS, since support for other platforms isn't currently available. Another issue that some developer might have with Wikitude is the fact that it is paid software. Despite the fact that is possible to use most of the feature that the framework offer for free, this only is intended for testing and trial purposes, since the Augmented Reality view will have a water mark across the screen and the Wikitude Logo is displayed on the lower left corner of the screen. Since this work is not intended to

be made available at none of the applications markets/stores and the main test platforms already are iOS and Android, both of the previous mentions issues can be overlooked.

After the setting up the Wikitude plugin, it's necessary to implement the ARchitect World. ARchitect World is an HTML file that will be loaded over the camera stream when the augmented reality view is launched. This file, is simply put, what will "augment" the user reality. The ARchitect World doesn't need any specific setup beside the inclusion of the script that loads the Wikitude ARchitect API (as seen in the next code snippet), if the developer wants to use the augmented reality tools offered by Wikitude. This API offer several features like calculating distance between two points, add elements (text, images, shapes) to the screen attached to a set geographical location, add listeners to events related to elements on the screen, create animations, etc.

```
<script src="architect://architect.js"></script>
```

When creating an augmented reality view, the developer will not be inside the scope of the Sencha Touch framework, this means that the access to stores, views and controllers will not be possible from the moment the view is launched. This is a concern developer might have, especially if the interface of the application needs to be the same throughout the application, since Sencha Touch offers an interface to create visual elements that gives us a layer of abstraction when using HTML elements, which will not be available when creating the interface of the ARchitect World. During the development of this work, it was not found an easy solution to this problem, however since the ARchitect World is a regular HTML5 file, the developers will have at their disposal all the tools they already have available when developing HTML webpages, which among other features include the use of external CSS and JavaScript files.

Despite exiting the scope of the Sencha Touch framework, Wikitude give the developer a possibility to configure callbacks that will be called within the ARchitect World and will be handled by a previously defined function within the JavaScript that launched the augmented reality. This feature let the developers pass information between the main application and the augmented reality, which can be very useful if they want to use data from the augmented reality in rest of the application. Furthermore passing data from the main application to the augmented reality segment is also possible using the Wikitude plugin. This process is similar to the previous one since the developer will have

to implement a function that will handle the call from the main application, but contrary to when the function is called from within the AR World, to pass information to the main application, the developer can declare several function instead of declaring just one that have to handle differently the request depending on the parameters passed. To use this features it's necessary to call Wikitude plugin and state which function will be called and which parameters will be passed to it. Something that can make this process a little more complicated than it should is that all the information (function to be called and respective parameters) need to be put in a simple string.

6.3 Map View

The first view that is presented to the user when he launches the application is the map view. In this view a map is loaded through the use of the Google Maps JavaScript API, which is added to the project using the code bellow.

```
<script type="text/javascript"
      src="http://maps.google.com/maps/api/js?sensor=true">
</script>
```

This API offers a lot of map/geographical related features, which allow the developers to fully make use of what Google Maps has to offer. Among these features it stands out the ability to add custom markers to the maps, add custom message windows, add maps related event listeners and calculate path between points.

In this type of application, besides the map it's really important to know the GPS coordinates of the user position, so the application can display correctly the information related to its position. To have access to this information, it was used the HTML5 Geolocation API, as seen in the next code snippet. The API easily gets the user GPS coordinates, if it is possible to retrieve them, either using GPS sensor or Internet connectivity. Also, there is the possibility to add listeners to check if there are any updates to the user location, thus making the application more responsive to movements from the user.

```
navigator.geolocation.getCurrentPosition(successCallback);
```

As said above, the application needs to retrieve the POIs near the user, but if he wants to navigate within the map, all the points need to be available as soon as the map bounds change, avoiding unnecessary waiting periods when, for example, the maps is zoomed out. To give the user the best experience possible, the application will retrieve the points in two separate phases. The first phase is triggered after the map is loaded, this is the phase when the application will make a request to the Rama Web Service, which will retrieve the POIs in the area of the map visible to the user. To retrieve only the points within the bounds of the map, it is sent to the web service the coordinates of those bounds that were supplied by the Google Maps JavaScript API and retrieved as seen in the next code snippet.

```
var bounds = gmap.getMap().getBounds();
var ne = bounds.getNorthEast();
var sw = bounds.getSouthWest();
```

The second phase will start immediately after the first one ends. In this phase all the available POIs are retrieved from the web service, resorting to pre-determined bounds coordinates, which include all the POIs within them. Since there are a large number of POIs in the database and their number can always increase, this task can take some time to be executed. “Freezing” the interface while the task is being completed greatly reduces the user experience. To solve this issue, it was used an HTML5 feature called web workers, which enable a JavaScript to run in the background while interface keeps working normally [58]. In this case the web worker will be in charge of making the request to the web service, wait for the reply and then send it to the main JavaScript. After that, it is only necessary to parse the JSON received and save the data to a store that will hold the information while the application is active. This technique makes the longest stage of this task to run in the background, without affecting the usability of the application. Declaring the web worker, defining the function that handles the response and calling the web worker is done as seen in the next code snippet.

```
var webWorker = new Worker("path/to/webworkers/worker.js");
webWorker.onmessage = function (event) {
      handleWebWorker(event); //Function to Handle the Response
};
webWorker.postMessage(reqUrlAll); //URL of the Web Service is
                                  //passed to the web worker
```

The main functionality of the map view is to present to the user the POIs in Aveiro in an easy way to understand and have a good overview of all of them around the city. This means that the view is very simple, but it has some extra functionality aimed to improve the user experience. One of these features is the use of web storage. Web storage allow the application to save data offline (with local storage) so it isn't totally dependent of Internet connectivity. In the context of this application, this is especially useful because some users might use the application indoors where they have Internet connection and check the POI they are searching, but when they go outside to said POI they might lose connectivity, but will nevertheless be able to use the application, using the information saved when they were online. The Rama application uses web storage to save two JSON strings that contains the information about the POIs that were displayed on the map (exemplified in the code snippet bellow) and POIs retrieved while using the augmented reality view.

```
localStorage.setItem("mapPois",JSON.stringify(POIS));
```

Since this application is intended to be more robust and use more features that not only the Sencha Touch but also CSS3 and HTML5 offer, Rama makes use of media queries. Media queries are a CSS feature that enables the developer to set a group of CSS rules, which will be applied to the visual elements, depending on conditions defined by the developer. In this type of applications (mobile), the conditions are mainly related to the size and resolution of the smartphones screens. For example, the images displayed in a screen with 480px height will be different from the images displayed in a screen with 568px height (these are iPhone measures). This is very important to assure a responsive design, which will fit in as many screens as conditions defined by the developer. And since Sencha Touch was the framework chosen to develop this application, media queries being compatible with SASS is very helpful. The next code snippet shows how to apply CSS media queries using to two different screen dimensions.

The Figures 6-1, 6-2 and 6-3 display the interface of the application in both iOS and Android. The interface is very simple and is mainly focused in displayed the map as large as possible to the user, making it easy to navigate. Another feature that was implemented was the detection of orientation change, which is demonstrated in Figures 6-1 and 6-2 (although in the next figures is not displayed, the iOS application also add the same behaviour).

```
/*Media Queries used in iOS Devices*/
@media only screen
and (min-device-width : 320px) and (max-device-width : 568px)
{
      /*Specific Configuration*/
}
@media only screen
and (min-device-width : 320px) and (max-device-width : 480px)
{
      /*Specific Configuration*/
}
```

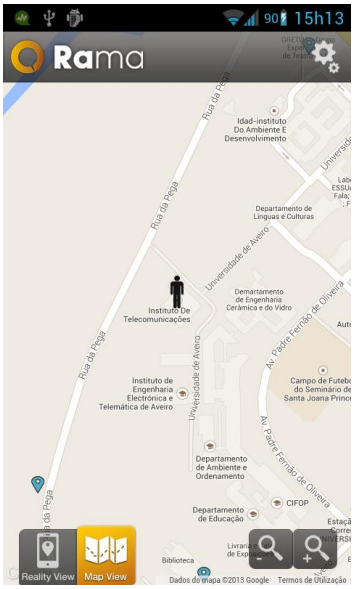

Figure 6-1 Rama Map View (Portrait) developed with Sencha Touch deployed in Android

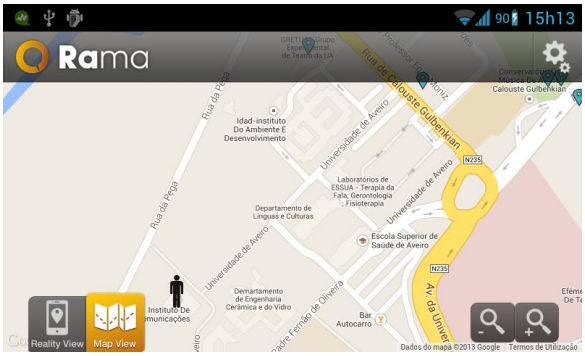

Figure 6-2 Rama Map View (Landscape) developed with Sencha Touch deployed in Android

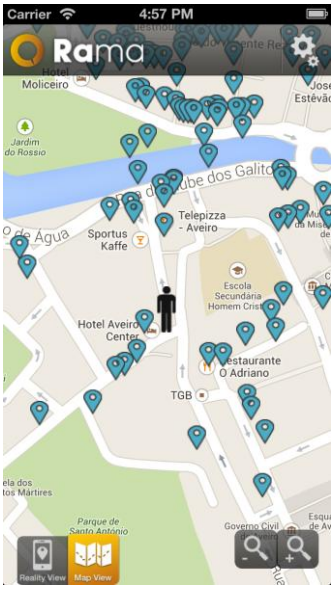

Figure 6-3 Rama Map View (Portrait) developed with Sencha Touch deployed in iOS

6.4 Augmented Reality View

As explained above to use augmented reality in a hybrid application it is necessary to resort to plugins and in this case Wikitude was the one chosen. To start the augmented reality view (after the AR World has been implemented), the developers need to call the Wikitude plugin, use the load world function and pass the local path of the AR World file, as seen in the next code snippet.

```
WikitudePlugin.loadARchitectWorld("path/to/ARWorld/ARWorld.html")
```

A small trick used in this specific application, was to force a location update from the smartphone, after the augmented view is already displayed to the user. This is necessary because Wikitude needs to identify the current position of the user, so it can post the marker on the screen related to the coordinates. The application could let the smartphone update the position by itself, but since this can take some time, forcing a small update by calling a function on the main JavaScript of the application, was found to be the best solution, in order to avoid having periods where the screen would not show any point. After the main JavaScript gets the updated location it calls a Wikitude function that let the developer set the current coordinates of the user, inside the AR World environment, as exemplified in the code bellow.

```
WikitudePlugin.setLocation(currentLocation);
```

The main objective of the augmented reality view is to show the POIs to the user, in a way that is easy to navigate in the "real world" and have a sense were they are relative to our current location. However, when displaying information in an augmented reality environment it's very important to try not cluster the screen too much and for this reason when the augmented reality is launched, the web service used to get the POIs is different from the one used in the map view. This second web service only needs the user current location and will return a maximum of 20 points to the application ordered by distance and in a 400 meters radius. However, not every point returned by this initial call will be displayed on the screen, because the application will only display points closer than 300 meters from the user current location. The resulting interface and POIs markers are displayed as seen in Figure 6-4.

To ensure that the view is regularly updated, it was implemented a location listener in the AR World, that when triggered will verify if the new coordinates are 200 meters apart from the last saved ones and if so a web worker will be called that will access the web service that retrieve more points near the user (as seen in the next code snippet). Some points will be replicate, but the application will only save new points. Besides this, the listeners will verify if the points currently displayed on the screen are still closer than 300 meters from the current coordinates and if they aren't they will be removed from the screen.

```
var webWorker = new Worker("path/to/webworkers/ARworker.js");
webWorker.onmessage = function (event) {
      handleARWebWorker(event); //Function that handles the web
                                //worker result
};
webWorker.postMessage(reqUrl); //Is passed the  url of the web
                               //service to the web worker
```

Every point displayed on the screen has many details about them that can't be present on the screen all the time however, this data is available by clicking on the marker of that POI. When a marker is tapped, a small tab in the bottom of the screen will be displayed with the POI name, type, current distance to the user and a button that let the user see even more details about the local (phone number, email, etc.) as seen in the Figure

6-5. To close this details tab, the user only needs to tap on the screen. The listeners of the tap event on the POI marker it's set using the trigger option in the marker element of the Wikitude ARchitect API and the tap on the screen event is set using a context event listener also provided by the Wikitude ARchitect API.

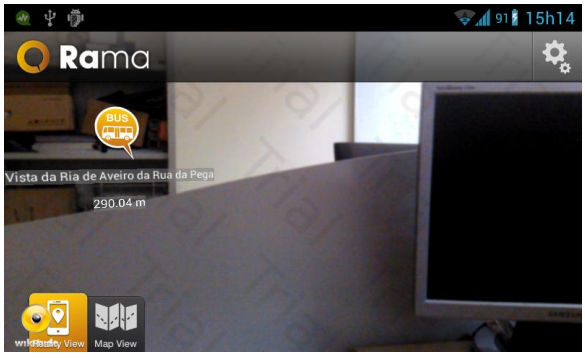

Figure 6-4 Rama Augmented Reality View developed with Sencha Touch deployed in Android

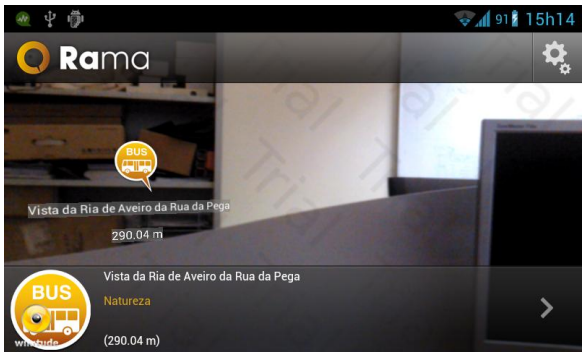

Figure 6-5 Rama Augmented Reality View with PoI details panel open, developed with Sencha Touch deployed in Android

6.5 Android vs. iOS Implementation

One of the main focuses of this work was to evaluate the differences in implementations when changing platform and how complex the changes needed would be. In the case of the Rama application the platforms that were fully supported were only Android and iOS, because the restriction at the moment that the Wikitude Phonegap plugin present. This was not a major issue since during the development of this dissertation those 2 platforms owned nearly 90% of the world smartphone market share.

Concerning the map view, the changes made between the Android and iOS application were all related to the interface of the view. This is due to the difference in screen size between Android and iOS devices. Also the newest IPhones have a retina display which was something that was taken in account. To assure that the application have a similar interface in both platforms, independently of the screen size, media queries were used in the SASS files that generated the main CSS file. In the Android SASS file it was taken into account the same categories of screen resolutions that native Android applications do: MDPI, HDPI and XHDPI. In the iOS SASS file were created categories to every screen resolution IPhone devices have at the moment (only 2). It was also taken into account if the screen uses retina technology. One important thing to notice is that the resolutions here described are measured in CSS pixels and not in real pixels, because the files that will manipulate the interface are written in CSS.

As for the augmented reality view, the implementation still was the same on both platforms, with the exception of the setup necessary to have Wikitude run correctly on each of them. The setup in the Android application was rather simple and the application worked like it was intended. On the other hand, in iOS, despite the fact that the sample project worked exactly as it was supposed, the Rama application was unable to launch the augmented reality view. This was unfortunate, since previously to this every other feature worked exactly the same way in iOS and Android, without the necessity to change almost any code. One possibility about the source of this problem is an error in the setup of the Wikitude plugin in iOS and its compatibility with the Phonegap version used. The problem being related to the implementation of the AR World or the augmented reality view was disregarded, since every other function so far, functioned as intended.

6.6 Windows Phone and Desktop Browser

As in the SelfCare application, after the development process was almost concluded, it was found interesting to see how Rama would perform in a Windows Phone 8 platform. Since this application was more complex and the fact that Wikitude did not give support at the time to Windows Phone, the only real test made was on the map view of the application.

The interface in the Windows Phone 8 simulator was very similar to the one in Android and iOS and the request to get the POI was made successfully, however the interface was unresponsive, more specifically, none of the buttons on the screen respond to a touch event. Although the problem was not explored, its origin may be associated with the use of a Google Maps, which cover the entire smartphone screen, since in the previous application, no problem were found, related to the button and their events. As seen in Figure 6-6 the interface maintained most of its features (the zoom button were the only interface feature that was different).

During the development process, sometimes it was very useful to test the application with a faster method than deploying a new version through the Sencha Cmd, because sometimes the changes made to the code were relatively small. Since this dissertation aimed at using web technologies, the code developed was could also be run on a desktop browser. To test this, the index.html resulting from the build done with Sencha Cmd was hosted online (Dropbox) and then accessed through its public link. The results were very satisfactory since the UI was still the same as in the mobile application, as seen in the Figure 6-7 (thanks to CSS media queries) and every functionality (apart from the augmented reality) worked as intended.

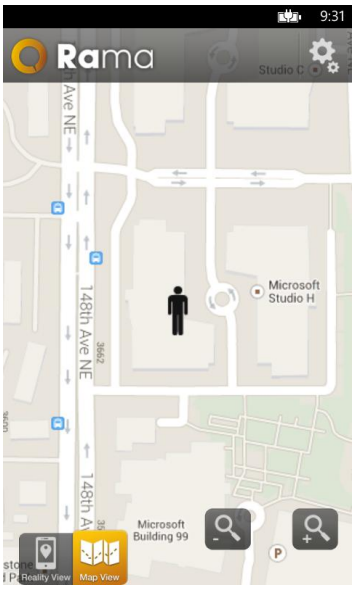

Figure 6-6 Rama Map View (Portrait) developed with Sencha Touch deployed in Windows Phone 8

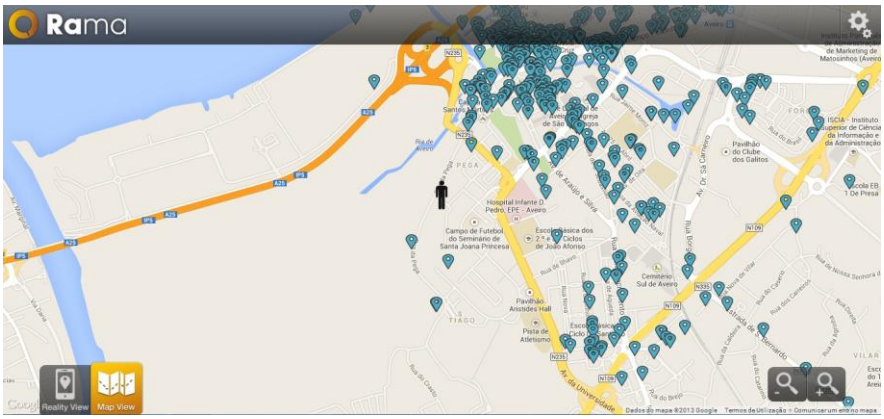

Figure 6-7 Rama Map View developed with Sencha Touch deployed in Desktop Browser (Google Chrome)

6.7 Conclusions

After the development of this application it is possible to see that HTML5 already has the features necessary to make more complex hybrid applications. Despite the fact that the support for augmented reality and some others more advanced features is not currently present, many plugins are already available to help developers implement more complex application. In this case the access to the smartphone camera given by Wikitude is very important in the Rama application and the augmented reality features were also very useful and easy to use.

In both platforms the application presented a good performance and although the native application seemed a little faster in term of transitions and animations, this was already to be expected.

		Memory	Storage
Rama	Sencha - iOS	162,87 MB	15,2 MB
	Sencha - Android	97 MB	3,65 MB
	iOS		500 KB
	Android	37 MB	634 KB

Table 6-1 Performance measurements made with Rama HTML5 and Native application

Like in the SelfCare, the native application uses less storage space from the device (the data about the memory usage was not possible to be retrieved during the development

of the project). It is also possible to see that the Android application is much smaller and use almost half the memory of the iOS application, this happens due to the lack of optimization of the iOS project, not the HTML5 code. Although the data about the native Android code are presented, they are not very relevant since the native Android application was not finished and was in fact dropped by PT Inovação at the time of the development of this project.

Something important about this application was the implementation of performance improvement techniques like web workers and local storage. These kinds of HTML5 features are very important to give the user a much better experience, closer to what a native application can offer.

Other improvement in this application compared to SelfCare, was the implementation of features like media queries, which are very useful in offering the user a flexible and responsive design of the interface. This responsiveness was mainly improved, thanks to CSS3 and Sencha Touch itself. Since this application had a greater focus in recreating the original Rama interface, the use of the same resources (images, logos, etc.) was crucial to achieve this and both CSS3 and ST were very capable of fulfilling the needed task.

As demonstrated through the desktop browser version of Rama, one of the most important aspects that HTML5 bring to the table, when talking about smartphone applications is its versatility, since has seen above, these kinds of applications can easily be replicated in many platforms and even be used in devices very different from the ones original intended. As it’s not uncommon to some businesses having a website and a smartphone application that accomplish the same goal (like YouTube or Facebook), using HTML5 can be a great way to make this process faster to the development team, because the code used can be the same, with just some interface tweaks.

7. Final Conclusions and Future Work

7.1 Final Conclusions

This work focused on the development comparison of smartphone applications resorting to web technologies like HTML5, JavaScript and CSS3, which were presented as a possible alternative to native development.

To make a better use of the features offered by these technologies, it was used two JavaScript frameworks, focused in the development of smartphone application: Sencha Touch and jQuery Mobile. Despite having the same goals, these two frameworks are different in several aspects and therefore it was also very important, to compare them through the development of the same application two times, using each of the frameworks. This analysis was then instrumental in choosing which framework was used in the development of other application, with more complex features.

The application used as a starting point to the comparison between the two JavaScript frameworks was very simple, however it had some very key features that needed to be taken into account when developing smartphone applications. During the development process these features were implemented and then classified according to some predetermined evaluation criteria. These criteria were however only a portion of the evaluation conducted, since performance indicators and community support were also taken into account.

The final evaluation conducted resulted in Sencha Touch being selected to develop the next application. The reasons behind the decision to use Sencha Touch were mainly the fact that it promotes the use of a MVC pattern and that it provides tools that make the deployment process much easier and structured, without making it much more complex. Although Sencha Touch was the framework chosen, jQuery Mobile also presented advantages like its learning easiness, the great amount of support from the development community and better results, when talking about performance indicators.

The second application developed was more complex than the previous one, since it needed to use external API and plugins in conjunction with Sencha Touch. This application focused on displaying relevant points (Point of Interest) in the city of Aveiro

and to accomplish that, the application had to make use of maps and augmented reality. Since Sencha Touch doesn't provide direct support to the use of maps and, at the moment, no support at all to the use of augmented reality, it was necessary to use external plugins and API. In the case of the maps, as mentioned above, Sencha Touch offer some support, but that support is subject to the use of Google Maps JavaScript API, which was the option taken. To implement augmented reality features, it was necessary to use a Phonegap plugin called Wikitude. This plugin offers image recognition and augmented reality features to the developers, but needs to be used in conjunction with Phonegap to be implemented in hybrid smartphone applications.

The development of this application also had the goal of using some key HTML5 features, which could help enhance the performance of the smartphone applications. In this case in particular, web worker, offline storage, integration with CSS3 and Geolocation API were the features explored and all were implemented with great success, contributing vastly in providing a cleaner (visually) and faster application.

The initial development and testing was done in an Android smartphone, however it was very important to the dissertation, to test the application in iOS as well. Since this application had some external resources besides Sencha Touch, the success from the cross platform application also depended on how Wikitude and the Google Maps JavaScript API behave on different platforms. The tests on the different platform were very successful since all the features had a very similar behaviour in Android and iOS, without the need to change any code, which is one of the key objectives of this kind of applications. However, not everything worked as planned, since the augmented reality features in iOS were not successfully implemented. This setback was probably caused by an incorrect setup of the Wikitude plugin in iOS or an incompatibility between the Phonegap version used in iOS and the Wikitude Phonegap plugin.

Performance wise, as expected the natives applications are still the ones that perform better in every field analysed in this dissertation. It is very important to understand that native application are optimized to the platform where they are running and therefore memory usage and storage space will be smaller. Between the two frameworks, jQuery Mobile showed a better performance in the Self Care application however, the better performance was not enough to make this framework the chosen one to develop Rama,

since the development tools offered to the developers were considered a better asset when creating a hybrid application.

After all the development, performance measurements, frameworks analysis and cross platform testing, what can be concluded from this work is that web technologies are already a viable alternative to native development when talking about smartphone applications. However, this conclusion must be looked with caution, because hybrid applications aren't now and probably will never be, as powerful as native applications if we look purely at performance numbers. But this doesn't mean that hybrid applications can't have a large market, especially in applications that don't require a lot of processing power and interface manipulation and that need to be available in many different platforms in a very short amount of time. Also, with the constant evolution in web technologies and smartphones browser, the features made available by hybrid application tend to grow.

7.2 Future Work

This dissertation focus on the study of Sencha Touch and jQuery Mobile, but there are many more JavaScript frameworks also focused on mobile development, which would be interesting to analyse and therefore compare with the ones already mentioned. This work could include the development of applications already developed using JQM and ST or through the development of completely new applications that could focus on some other features, not addressed in the dissertation.

Another way to further develop this topic would be to continue analysing the possibilities offered by both Sencha Touch and jQuery Mobile. This includes the development of the Rama application using jQuery Mobile and then work on some other applications that could focus on different features, not addressed by either Self Care or Rama.

Finally, this work could also be deepened through the deployment of the applications developed in other platforms. This dissertation mainly focuses on the behaviour of hybrid application in Android and iOS, since these two platforms are the current leaders of the smartphone market. This work also makes a minor reference to Windows Phone, since it's the platform with the biggest projected growth in the following years. However, to really analyse how reliable web technologies are, it would be very important to test them in different platforms like Symbian, Blackberry or even Bada.

References

[1] W3Schools, “HTML5 Introduction.” [Online]. Available: http://www.w3schools.com/html/html5_intro.asp. [Accessed: 09-May-2013].

[2] Vision Mobile, “Develor Economics 2013 - Developer Tools: The Foundations of the App Economy,” pp. 1–61, 2013.

[3] IDC, “Android and iOS Combine for 91.1% of the Worldwide Smartphone OS Market in 4Q12 and 87.6% for the Year, According to IDC - prUS23946013.” [Online]. Available: http://www.idc.com/getdoc.jsp?containerId=prUS23946013#.UWwXKrXvvzx. [Accessed: 09-May-2013].

[4] Gartner, “Gartner Says Worldwide Smartphone Sales Reached Its Lowest Growth Rate With 3.7 Per Cent Increase in Fourth Quarter of 2008.” [Online]. Available: http://www.gartner.com/newsroom/id/910112. [Accessed: 09-May-2013].

[5] Gartner, “Gartner Says Worldwide Mobile Phone Sales Declined 1.7 Percent in 2012.” [Online]. Available: http://www.gartner.com/newsroom/id/2335616. [Accessed: 09-May-2013].

[6] Online Publishers Association and I. Frank N. Magid Associates, “A Portrait of Today’s Smartphone User,” no. August, 2012.

[7] HTC Corporation, “HTC - Press - 2008 - T-Mobile Unveils the T-Mobile G1 — the First Phone Powered by Android.” [Online]. Available: http://web.archive.org/web/20110712230204/http:/www.htc.com/www/press.aspx?id=66338&lang=1033. [Accessed: 09-May-2013].

[8] StatCounter, “Top 9 Mobile Browsers from Apr 2012 to Apr 2013 | StatCounter Global Stats.” [Online]. Available: http://gs.statcounter.com/#mobile_browser-ww-monthly-201204-201304. [Accessed: 31-May-2013].

[9] D. Seven, “What is a Hybrid Mobile App?” [Online]. Available: http://www.icenium.com/blog/icenium-team-blog/2012/06/14/what-is-a-hybrid-mobile-app-. [Accessed: 31-May-2013].

[10] inc. Salesforce.com, “Native, HTML5, or Hybrid: Understanding Your Mobile Application Development Options.” [Online]. Available: http://wiki.developerforce.com/page/Native,_HTML5,_or_Hybrid:_Understanding_Your_Mobile_Application_Development_Options. [Accessed: 31-May-2013].

[11] R. Berjon, “HTML5 — Smile, it’s a Snapshot! - W3C Blog.” [Online]. Available: http://www.w3.org/QA/2012/12/html5_smile_its_a_snapshot.html. [Accessed: 13-May-2013].

[12] WHATWG, “1 Introduction — HTML Standard.” [Online]. Available: http://www.whatwg.org/specs/web-apps/current-work/multipage/introduction.html#history-1. [Accessed: 13-May-2013].

[13] W3Schools, “CSS3 Introduction.” [Online]. Available: http://www.w3schools.com/css3/css3_intro.asp. [Accessed: 13-May-2013].

[14] The jQuery Foundation, “jQuery Mobile | jQuery Mobile.” [Online]. Available: http://jquerymobile.com/.

[15] Sencha Inc., “HTML5 Mobile App Development Framework. Download Sencha Touch Free. | Sencha Touch | Products | Sencha.” [Online]. Available: http://www.sencha.com/products/touch/. [Accessed: 10-May-2013].

[16] The Dojo Foundation, “Dojo Mobile - The Dojo Toolkit.” [Online]. Available: http://dojotoolkit.org/features/mobile. [Accessed: 08-Oct-2013].

[17] T. Fuchs, “Zepto.js: the aerogel-weight jQuery-compatible JavaScript library.” [Online]. Available: http://zeptojs.com/. [Accessed: 08-Oct-2013].

[18] XUI Contributers, “xui.js - a simple javascript library for building mobile web applications.” [Online]. Available: http://xuijs.com/. [Accessed: 08-Oct-2013].

[19] The Dojo Foundation, “Wink toolkit - A mobile JavaScript framework to build great webapps.” [Online]. Available: http://www.winktoolkit.org/. [Accessed: 08-Oct-2013].

[20] Sencha Inc., “Fastbook.” [Online]. Available: http://fb.html5isready.com/. [Accessed: 08-Oct-2013].

[21] J. Resig, “jQuery Mobile Alpha 1 Released | jQuery Mobile.” [Online]. Available: http://jquerymobile.com/blog/2010/10/16/jquery-mobile-alpha-1-released/. [Accessed: 08-May-2013].

[22] The jQuery Foundation, “Announcing jQuery Mobile 1.3.1 | jQuery Mobile.” [Online]. Available: http://jquerymobile.com/blog/2013/04/10/announcing-jquery-mobile-1-3-1/. [Accessed: 08-May-2013].

[23] Dmethvin, “jQuery Licensing Changes | Official jQuery Blog.” [Online]. Available: http://blog.jquery.com/2012/09/10/jquery-licensing-changes/. [Accessed: 08-May-2013].

[24] The jQuery Foundation, “Original Graded Browser Matrix | jQuery Mobile.” [Online]. Available: http://jquerymobile.com/original-graded-browser-matrix/. [Accessed: 08-May-2013].

[25] The jQuery Foundation, “Mobile Graded Browser Support | jQuery Mobile.” [Online]. Available: http://jquerymobile.com/gbs/. [Accessed: 08-May-2013].

[26] The jQuery Foundation, “jQuery Mobile Docs - Quick start.” [Online]. Available: http://jquerymobile.com/demos/1.3.0-beta.1/docs/about/getting-started.html. [Accessed: 08-May-2013].

[27] The jQuery Foundation, “Introduction.” [Online]. Available: http://jquerymobile.com/demos/1.3.0-beta.1/docs/about/intro.html. [Accessed: 09-May-2013].

[28] The jQuery Foundation, “jQuery Mobile Docs - Features.” [Online]. Available: http://jquerymobile.com/demos/1.3.0-beta.1/docs/about/features.html. [Accessed: 09-May-2013].

[29] The jQuery Foundation, “ThemeRoller | jQuery Mobile.” [Online]. Available: http://jquerymobile.com/themeroller/. [Accessed: 09-May-2013].

[30] The jQuery Foundation, “jQuery Mobile Framework - Themes.” [Online]. Available: http://jquerymobile.com/demos/1.3.0-beta.1/docs/api/themes.html. [Accessed: 09-May-2013].

[31] The jQuery Foundation, “jQuery Mobile Docs - Events.” [Online]. Available: http://jquerymobile.com/demos/1.3.0-beta.1/docs/api/events.html. [Accessed: 09-May-2013].

[32] The jQuery Foundation, “jQuery Mobile Docs - Methods.” [Online]. Available: http://jquerymobile.com/demos/1.3.0-beta.1/docs/api/methods.html. [Accessed: 09-May-2013].

[33] The jQuery Foundation, “jQuery Mobile Docs - Data Attribute Reference.” [Online]. Available: http://jquerymobile.com/demos/1.3.0-beta.1/docs/api/data-attributes.html. [Accessed: 09-May-2013].

[34] Sencha Inc., “Mobile HTML5 Framework - Features of Sencha Touch | Features | Sencha Touch | Products | Sencha.” [Online]. Available: http://www.sencha.com/products/touch/features/. [Accessed: 10-May-2013].

[35] A. Elias, “Ext JS + jQTouch + Raphaël = Sencha | Blog | Sencha.” [Online]. Available: http://www.sencha.com/blog/ext-js-jqtouch-raphael-sencha. [Accessed: 10-May-2013].

[36] A. Elias, “Sencha Touch 1.0 Ships - Now Free! | Blog | Sencha.” [Online]. Available: http://www.sencha.com/blog/sencha-touch-10-ships-now-free/. [Accessed: 10-May-2013].

[37] Sencha Inc., “Licensing | Sencha Touch | Products | Sencha.” [Online]. Available: http://www.sencha.com/products/touch/license/. [Accessed: 10-May-2013].

[38] A. Bansod, “Hello Sencha Touch 2.2 | Blog | Sencha.” [Online]. Available: http://www.sencha.com/blog/hello-sencha-touch-2-2/. [Accessed: 10-May-2013].

[39] Sencha Inc., “The Class System - Touch 2.2.0 - Sencha Docs.” [Online]. Available: http://docs.sencha.com/touch/2.2.0/#!/guide/class_system. [Accessed: 10-May-2013].

[40] Rj45, “Simple Example of MVC (Model View Controller) Design Pattern for Abstraction - CodeProject.” [Online]. Available:

http://www.codeproject.com/Articles/25057/Simple-Example-of-MVC-Model-View-Controller-Design. [Accessed: 10-May-2013].

[41] Sencha Inc., “All about Applications - Touch 2.2.0 - Sencha Docs.” [Online]. Available: http://docs.sencha.com/touch/2.2.0/#!/guide/apps_intro. [Accessed: 10-May-2013].

[42] E. Spencer, “Getting Started with Sencha Touch 2 | Learn | Sencha.” [Online]. Available: http://www.sencha.com/learn/getting-started-with-sencha-touch-2. [Accessed: 10-May-2013].

[43] Sencha Inc., “Introduction to Sencha Cmd - Touch 2.2.0 - Sencha Docs.” [Online]. Available: http://docs.sencha.com/touch/2.2.0/#!/guide/command. [Accessed: 10-May-2013].

[44] Sencha Inc., “Using Sencha Cmd with Sencha Touch - Touch 2.2.0 - Sencha Docs.” [Online]. Available: http://docs.sencha.com/touch/2.2.0/#!/guide/command_app. [Accessed: 13-May-2013].

[45] H. Catlin, N. Weizenbaum, and C. Eppstein, “Sass - Syntactically Awesome Stylesheets.” [Online]. Available: http://sass-lang.com/. [Accessed: 13-May-2013].

[46] C. M. Eppstein, “Compass Home | Compass Documentation.” [Online]. Available: http://compass-style.org/. [Accessed: 13-May-2013].

[47] Sencha Inc., “Touch 2.2.0 - Sencha Docs.” [Online]. Available: http://docs.sencha.com/touch/2.2.0/. [Accessed: 08-Oct-2013].

[48] Adobe Systems Inc., “PhoneGap | PhoneGap Explained Visually.” [Online]. Available: http://phonegap.com/2012/05/02/phonegap-explained-visually/.

[49] The jQuery Foundation, “jQuery Mobile Docs - Forms.” [Online]. Available: http://jquerymobile.com/demos/1.3.0-beta.1/docs/forms/docs-forms.html. [Accessed: 16-May-2013].

[50] W3Schools, “HTML5 Input Types.” [Online]. Available: http://www.w3schools.com/html/html5_form_input_types.asp. [Accessed: 16-May-2013].

[51] The jQuery Foundation, “jQuery.ajax() | jQuery API Documentation.” [Online]. Available: http://api.jquery.com/jQuery.ajax/. [Accessed: 16-May-2013].

[52] The jQuery Foundation, “Headers - jQuery Mobile Demos.” [Online]. Available: http://view.jquerymobile.com/1.3.1/demos/widgets/headers/. [Accessed: 17-May-2013].

[53] Sencha Inc., “Ext.Container - Touch 2.2.0 - Sencha Docs.” [Online]. Available: http://docs.sencha.com/touch/2.2.0/#!/api/Ext.Container-method-animateActiveItem. [Accessed: 23-May-2013].

[54] M. Jackson, S. Crouch, and R. Baxter, “Software Evaluation : Criteria-based Assessment,” pp. 1–13, 2011.

[55] Yahoo! Inc., “Best Practices for Speeding Up Your Web Site.” [Online]. Available: http://developer.yahoo.com/performance/rules.html. [Accessed: 13-Sep-2013].

[56] Adobe Systems Inc., “PhoneGap | Supported Features.” [Online]. Available: http://phonegap.com/about/feature/. [Accessed: 13-Aug-2013].

[57] Wikitude GmhB, “PhoneGap Plugin - Wikitude SDK Documentation - Devzone.” [Online]. Available: http://developer.wikitude.com/documentation/phonegap. [Accessed: 13-Aug-2013].

[58] W3Schools, “HTML5 Web Workers.” [Online]. Available: http://www.w3schools.com/html/html5_webworkers.asp. [Accessed: 12-Aug-2013].