FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Cross-platform mobile development using Web Technologies

Diogo Costa

Mestrado Integrado em Engenharia Informática e Computação

Supervisor: Rui Maranhão (PhD)

July 10, 2013

Cross-platform mobile development using Web Technologies

Diogo Costa

Mestrado Integrado em Engenharia Informática e Computação

Approved in oral examination by the committee:

Chair: António Miguel Pontes Pimenta Monteiro (PhD)
External Examiner: José Maria Amaral Fernandes (PhD)
Supervisor: Rui Filipe Lima Maranhão de Abreu (PhD)

July 10, 2013

Abstract

There are many different mobile platforms nowadays that share the same applications, at least their name and brand. The differences between each of these mobile platforms make it very hard to build a single code base that runs in all platforms. However, efforts are being made to turn this into a possible scenario.

Since almost all major platforms provide a web engine that supports HTML5, CSS3 and Javascript, these technologies have become the base for most cross-platform development framework efforts. Using HTML5, Javascript and an application wrapper, such as PhoneGap, it is possible to make a web application that runs just like a native one. One of the major drawbacks of this approach is developing the user interface. Each platform has its user interface guidelines and designing an interface for each platform can be time consuming.

This dissertation addresses the implementation of framework that aims to simplify the development of cross-platform applications using HTML5, CSS3 and Javascript, including a library that auto-adjusts to the look and feel of the platform the application is running on. A demo application will also be described, showcasing the developed framework.

Keywords: Mobile, HTML5, CSS3, Javascript, cross-platform, PhoneGap, Meems

Resumo

Atualmente existem diferentes plataformas móveis que partilham as mesmas aplicações, ou pelo menos o mesmo nome e marca. As diferenças entre estas plataformas móveis tornam muito difícil a utilização do mesmo código fonte para a geração de executáveis para cada uma das plataformas. Contudo, estão a ser feitos esforços no sentido de tornar este cenário uma realidade.

Visto que as grandes plataformas móveis disponíveis atualmente suportam HTML5, CSS3 e Javascript, estas tecnologias tem vindo a tornar-se a base da maior parte dos esforços para criar uma *framework* para desenvolvimento de aplicações móveis multiplataforma. Através da utilização de HTML5, Javascript e um *wrapper* como o PhoneGap é possível criar uma aplicação web que se assemelha e se comporta como uma aplicação nativa. No entanto, um dos grandes entraves neste momento a este tipo de solução é o desenvolvimento da interface gráfica do utilizador. Cada plataforma tem os seus próprios "guidelines" gráficos e desenvolver uma interface por plataforma é uma tarefa que consome bastante tempo.

Esta dissertação endereça a implementação de uma *framework* que tem por objetivo a simplificação do desenvolvimento de aplicações móveis multiplataforma, recorrendo para isso às tecnologias HTML5, CSS3 e Javascript. A *framework* inclui uma biblioteca para desenvolvimento de interfaces gráficas do utilizador que ajustam a sua aparência e comportamento consoante a plataforma em que a aplicação está a ser executada. É descrita também uma aplicação de demonstração, cujo objetivo é mostrar como utilizar a *framework* desenvolvida e mostrar do que é capaz.

Keywords: aplicações, móveis, HTML5, CSS3, Javascript, multiplataforma, PhoneGap, Meems

Acknowledgements

I would like to say thank you to my family, particularly my grandparents, for supporting me during the past year, when I worked and studied at the same time. Their contribute was crucial for me to be able to finish this dissertation.

I would also like to thank my parents, for believing in me and their unconditional support, and my girlfriend Marisa for fighting so that we can both have a good life in the future.

At last, but by no means least, I would like to thank my supervisor Rui Maranhão, for accepting to guide me throughout this dissertation and providing his support and expertise, and my colleagues at Novabase for their friendship and support.

Thank you all.

"Anything can change, because the smartphone revolution is still in the early stages."

Tim Cook

Contents

List of Figures

LIST OF FIGURES

Abbreviations

ADT	Android Developer Tools
API	Application Programming Interface
CSS	Cascading Style Sheets
GPS	Global Positioning System
HTML	HyperText Markup Language
IDE	Integrated Development Environment
NDK	Native Development Kit
SDK	Software Development Kit
SPA	Single Page Application
UI	User Interface
W3C	World Wide Web Consortium
WHATWG	Web Hypertext Application Technology Working Group

Chapter 1

Introduction

Smartphones are changing the way we live our lives. Checking emails, listening to music, sending texts, video chatting, GPS navigation, watching movies or taking notes are just a small subset of the tasks most smartphone owners perform. The practicality of these small but powerful devices that fit in a pocket is attracting more people every day.

In 2007 and 2008, the two major mobile operating systems were released to the market: Apple's iOS and Google's Android. Together they power the largest group of smartphones, making them the most popular mobile operating systems. Others have appeared meanwhile, like Windows Phone, BlackBerry OS or Bada, each with relative success.

The success of smartphones has been also directly related to its application ecosystem. The ability to download third party applications allows users to power up their phones with new functionalities without being tied to the operating system's provider. This also created opportunities for companies and developers to profit by developing new applications. Many have taken this opportunity and today there are thousands of mobile applications available for download.

Developing for a mobile platform, however, has brought new challenges compared to the traditional development of desktop applications. Smartphones have different specifications and capabilities than desktop computers and require good planning and implementation to build an application that is easy to use and useful for the final users. Some guidelines and general coherence is therefore necessary to reduce fragmentation and provide users with an excellent user experience on their usually small-sized-screen smartphones. Every mobile operating system has its own development, graphics and user interface guidelines. Following these guidelines ensures that third-party applications integrate well with the rest of the platform.

However, this fragmentation also means that in order to build applications for several platforms, one must develop different applications, one per platform. The advent of HTML5 and its growing support in mobile browsers has put the development of cross-platform web applications as an alternative to native development. Web applications do have limitations, but with the increasing popularity of this approach, people have been turning to hybrid applications, mixing web and native code to produce cross-platform applications.

1.1 Context and Motivation

This dissertation was born from my personal necessity to build mobile applications that look native as fast as possible and without spending too much money. After a quick survey of the available tools in market, I wasn't satisfied with the fact that no good free/open-source solutions were available. PhoneGap looked like a great starting point, enabling the packaging of web applications for six different platforms, but, although there are many frameworks and libraries for the development of mobile web applications, none would provide a native look and feel according with the platform it was running on.

Developing mobile applications for different platforms requires different programming languages and tools. Using web technologies, one can use HTML5, CSS3 and Javascript to build applications that run across all major operating systems, thus saving time and money while producing more. This only applies to certain types of applications, of course. There are cases when a native application is not only desirable but required, like those that need to interface directly with the phone's hardware.

With this dissertation, I intend to develop a framework that help developers build their mobile applications faster and cheaper, but still looking good on each mobile operating system, focusing on Android and iOS at first.

1.2 Goals

The main goal of this project is to produce a framework that enables developers to build mobile web applications that look native on Android and iOS platforms using web technologies, namely HTML5, CSS3 and Javascript. The developed framework will use open-source and will also be available entirely as open-source, as a contribution to the community that has offered, and continues to offer, me so much, in terms of knowledge, support and free technology. This will also enable other people to contribute to the project, allowing it to grow and be useful to other people.

But first, a study over the existing and most popular technologies must be performed in order to identify their most attractive aspects and those they are lacking, so that they can be included in the framework. The necessities of the market and developers is also to be taken into account, so a survey must also take place.

The definition of the user interface must be extensible, allowing the developer to create its own component when necessary, and must adapt itself to the look&feel of the platform it is running on, allowing, however, the developer to override the default behaviour if needed.

The framework must use existing, approved and popular libraries and tools, like PhoneGap and require.js, to be built over good practices and to appeal to as much developers as possible.

After developing the base framework, a demonstration application will also be developed. This application, being for demonstration purposes only, will showcase the framework and how to use it. The application must run on Android, iOS and the most common web browsers. A server-side

component must also be developed in order to demonstrate how to build an application end-to-end using Javascript as the base programming language.

1.3 Dissertation's Structure

This dissertation has 5 main chapters: "State of the Art" (chapter 2), "Case Studies" (chapter 3), "Market Study and Analysis" (chapter 4), "The Meems Framework" (chapter 5) and "RSS Reader: Demo Application" (chapter 6).

After the introductory chapter (chapter 1), different approaches when developing for mobile devices will be shown in chapter 2, along with general information on the mobile operating systems available nowadays and how to web applications can be published in the stores of the vendors of the platform. Then, the main web technologies this dissertation focus on are presented, namely HTML5, CSS3 and Javascript. In this chapter, a subset of the existing frameworks, tools and libraries that enable developers to build web mobile applications will also be presented, along with a comparison between them.

In chapter 3, some interesting and famous mobile web applications are exposed, along with a short analysis of the patterns used in the user interface and general trivia about the development of each application. Some conclusions about developing for mobile and the advantages of using web technologies for such task are drawn.

Chapter 4 reveals the main results of a survey that was conducted in the context of this dissertation are presented and interpreted.

Chapter 5 describes the development of the main framework that was born from this dissertation. The goals and main concepts of the framework are discussed first, followed by a description of the architecture and functionalities developed. The difficulties and challenges in developing this solution are presented at the end of the chapter, together with conclusions about the development.

Besides the main framework, a demonstration application was also developed, as described in chapter 6. The requirements of the application, its user interface and architecture of both mobile client and back-end are described in this chapter. Some conclusions about the development experience are drawn.

Finally, chapter 7 closes this dissertation, with overall conclusions about the work developed and its goals, a personal opinion about the future of the world of mobile development and some ideas about what work could be developed in the future to improve the Meems framework.

Chapter 2

State of the Art

The development of mobile applications is a recent field and has been growing in the past few years. The appearance of more powerful smartphones and operating systems is causing this field to evolve and grow exponentially.

In this chapter, a brief history of recent mobile operating systems is presented, along with a description of the common approaches taken to develop third-party software to these platforms, with special focus on web technologies, particularly the HTML5 standard. Some mobile web application development frameworks and libraries are also presented and analyzed.

Finally, the results of the survey performed in the context of this dissertation are presented and interpreted, closing off this chapter with a few conclusions about the gaps this dissertation intends to fill.

2.1 Mobile Operating Systems

Smartphones have become an important tool in our life. They've become practical, powerful and extensible. The evolution in hardware and software is intertwined, better hardware enables better software, and better software makes the user want more, leading to better hardware.

In the past few years, a revolution in the mobile fields has took place. The advent of modern operating systems like Android and iOS has changed the way people use smartphones.

2007 and 2008 were the years Google's Android and Apple's iOS were presented to the world. The iPhone, by Apple, was first revealed in January 2007, running the "iPhone OS", re-branded in 2010 to iOS. Later, in September 2008, the first phone running Google's Android operating system was presented: the T-Mobile G1.

The iPhone started this revolution, bringing to users a powerful phone designed to do more than just communicate. It included multitasking and graphics capabilities and sported a multi-touch screen, accelerometers, camera and other hardware components that contributed to a rich user experience. People were very receptive to this device, as sales have proven: 6.1 million 1st generation iPhones were sold over five quarters.[App09]

The T-Mobile G1, also known as HTC Dream, the first phone powered by Android, was quite as successful as the iPhone, but Android has been evolving and gaining lovers all over the world.

In the first quarter of 2013, Android was the leader operating system in the market, with a 75% market share, followed by iOS's 17.3% and Windows Phone's 3.2%.[IDC13]

One of the differentiating aspects that contributed to the success of these platforms over the dominant operating systems when they appeared, was the creation of a whole ecosystem around applications. Both Apple and Google created an online store to enable third-party vendors to distribute their applications and to allow users to download applications from a common and safe place. The possibility to extend the native platform with new applications meant that whenever an user feels a feature is missing, he can complement the platform with a third party application. This way, operating system vendors can focus on the platform, other companies and indie developers can build and distribute their own applications and make money, and the final user gets the benefits of a powerful multitasking device.

The next section describes how developers can build applications for these platforms.

2.2 Approaches to Mobile Development

When developing for mobile platforms, there are three possible approaches: native, web and hybrid.

The native approach consists of using solely the platform's software development toolkit to build an application that is compiled to the platform's native language.

The web approach consists of using web technology, such as HTML5, Javascript and CSS, to build a web application, accessible through the smartphone's browser and that the user can bookmark for later access.

Finally, the hybrid approach tries to combine the native and web approaches seamlessly, providing the final user with an application that seems native while allowing the developer to mix native and web technologies.

2.2.1 Native

Most, if not all, modern mobile operating systems provide developers with the necessary tools to develop applications that can later be installed by users to extend the smartphone capabilities. Each vendor provides developers with different tools, but the basics include a compiler and a debugger. These tools enable the developer to build native applications, with access to all the capabilities of the smartphone, such as camera, GPS, sensors, hardware acceleration, etc. Taking advantage of this capabilities, the developer can build very interesting applications that integrate with the platform and provide an excellent user experience.

For instance, two of the most prominent platforms, Android and iOS, provide several tools for application development.

Android tools allow developers to build applications for Android on Windows, Linux and Mac platforms, using the Android SDK Tools, an Android system image emulator, platform-specific tools and, optionally, the ADT plugin for the Eclipse IDE. Applications developed with this tools will be compiled to bytecode and run over Android's optimized virtual machine named Dalvik.[Goo13b] For developing applications that need extra performance, the Android NDK is available. It is a toolset that allows developers to use C or C++ to build native code.[Goo13a]

For developing applications for the iOS platform, which means targeting the iPhone, iPod and iPad devices, it is mandatory to use Xcode, Apple's integrated development environment, which already includes the iOS SDK. It is only available for Mac platforms, a restriction that makes developing for this platform an expensive investment if the developer doesn't own a Mac device already. Application logic is specified using Objective-C and a screen and flow designer is included in the toolkit. Applications can be tested by running in the included simulator. [App13]

2.2.2 Web

The idea of using a single code base for targeting several platforms has been the main motivation for using web technologies such as HTML5, CSS3 and Javascript to develop applications. And with access to the Internet and navigating through websites being a common functionality on smartphones, web applications for mobile devices have become popular.

Recent smartphones include powerful Web browsers and Javascript engines, capable of rendering complex websites and with a very good support of the HTML5 and CSS3 standards. The default browsers in Android, iOS and BlackBerry 6+ use the WebKit layout engine to render web pages. This toolkit is well known for its HTML5 and CSS3 support. Other layout engines are also used by other platforms: Internet Explorer Mobile uses Trident, Opera Mobile and Mini use Presto, Firefox Mobile uses Gecko. Although the standards are the same, their implementation may vary from an engine to another. This changes may mean that the same page will be rendered a bit different in distinct browsers. Therefore, in order to provide a cross-platform experience, the developer must be aware of the differences between each engine and try to accommodate them.

Besides the layout engine, most mobile browsers also incorporate a Javascript engine, responsible for handling all the scripted logic of the web pages. There are also different engines, which can cause different behaviour, even in the same platform. For instance, the default browsers of different smartphones with the Android operating system, known for its fragmentation, can use the JavascriptCore or Google's V8 Javascript engines, depending on the capabilities of the smartphone and its vendor. Although the programming language is the same, the performance of each engine is different, and this can cause different behaviours in different browsers, sometimes compromising the user experience.

To deal with these differences, developers often use code libraries and frameworks that provide an abstraction layer and deal with the particularities of each platform, speeding up the development task and ensuring a consistent user experience.

2.2.3 Hybrid

The downside of web applications is that the user does not perceive them as native, as they must run inside the browser. Another downside is the access to the capabilities of the phone that are not exposed to the web application by the Javascript engine, such as the accelerometers or access to the file system.

To circumvent this situation, a thin native wrapper can be used to hold the web application. This wrapper is a native application that is customized with the application's branding, such as its name and icons, and, when opened, uses a web browser view to show the web application. It can also be used to extend the Javascript API that is exposed to the application.

A very famous example of this is PhoneGap. PhoneGap, now owned by Adobe, "is an open source solution for building cross-platform mobile applications with standards-based Web technologies like HTML, Javascript, CSS". PhoneGap supports several platforms (iPhone (3G and newer), Android, Blackberry OS, WebOS, Windows Phone 7, Symbian and Bada) and exposes through Javascript many APIs to access the smartphone's features: accelerometer, camera, compass, contacts, file, geolocation, media, network, notifications and storage. And its functionality can be extended through the plugin system.

Another solution that enables the use of web applications as native ones is appMobi. It provides developers with an HTML5 IDE, device emulation, debugging tools, game acceleration and build services for iOS, Android and other platforms. AppMobi Cloud Services extends HTML5 with features like in-app purchasing, push messaging, analytics, authentication and more.

Other solutions take this approach a step further. Appcelerator's Titanium Mobile allows the development of native applications using Web technologies the same way PhoneGap does, but it exposes a Javascript API that allows the creation of native UI components. So, the developer can choose either to design the app with HTML and CSS or to use the API to build a native UI.

2.2.4 Publishing applications

After developing an application, the developer can distribute it in the vendor's application store, for instance, Google Play for Android applications, Apple Store for iOS applications or Window Phone Apps+Games Store for Windows Phone applications. Each store has its own publishing policy, but usually a fee must be paid to publish applications in these stores, be it one-time only (Google Play) or yearly (iOS App Store). These policies can also restrict which applications can be published, based on their look and feel, if they are native or not, if they add new functionalities to existing applications and if they are compliant with the platform's guidelines. [Wik12][Goo13c]

Web applications cannot apply for publishing in some application stores, most notably in Google Play and Apple Store. These stores require applications to be native, or at least, to look like a native application. This is one of the most frequent uses of webview containers like PhoneGap: to allow web applications to be published to application stores, targeting several platforms with a single code-base.

2.3 Web technologies

Developing mobile applications using web technologies can bring great benefits to the developers in terms of cross-platform support and cost reduction. The promise of "write once, run everywhere" is very tempting and is what motivates the use of HTML5, CSS3 and Javascript for building mobile applications.

In this section, we focus on how and why HTML5 appeared and how useful it can be to develop cross-platform mobile applications.

2.3.1 The appearance of HTML5

The HTML5 standard began to be written in 2004 by the Web Hypertext Application Technology Working Group (WHATWG), while the World Wide Web Consortium (W3C) was working on XHTML 2.0. This standard was born from the disagreement between browser vendors and the W3C on how the Web could be fixed. XHTML 2.0 was too strict and wasn't backwards compatible with HTML, making it harsh for developers to build cross-browser websites. In 2008, WHATWG released to the public the first draft of the specification and, despite not being finished at the time, parts of HTML5 had already been implemented in web browsers. Later, in 2009, the W3C decided to drop the development of XHTML 2.0 and teamed up with the WHATWG to help developing HTML5.

HTML5 is the evolution of HTML4, a standard that adds many new functionalities and APIs to web browsers, trying to supersede external plugins like Flash, Silverlight and Java. These include audio and video support, scalable vector graphics, drag-and-drop, history management, web storage, geo-location and many others.

Enabling sites to store data larger than a cookie in the client's web browser is also a functionality included in the standard. It enables the developer to cache files, images or data manually. This is a subject that browser vendors have disagreed on, leading to four different APIs: web storage, web SQL database, IndexedDB and filesystem.[Mah13] Their support is very different, but hopefully this will change in the future. Besides these method for storing data, HTML5 also provide offline caching, where the resources of the web site can be cached and used for running it when no connectivity exists. This is very useful. Take, for instance, the Gmail Offline[1] web application. It allows the user to connect with his Gmail account and download his emails to the browser's offline storage and the application itself to the offline cache. This way, when the internet connection is down, the user is still able to open the application and view the emails he downloaded previously.

Browsers are not for sharing only text anymore. Loads of videos and audio are downloaded each day, especially since the appearance of video sharing sites like YouTube. Have gained such importance, the HTML5 standard introduces native support for reproducing video and audio without external plugins, like Adobe Flash, which are often missing in mobile. The supported formats are not yet specified, but "Ogg Theora" and "H.264" are the most likely to be supported since

[1] http://goo.gl/T4RYW

they are both widely used open standards. At the moment, Internet Explorer and Safari support "H.264", while Firefox and Opera support "Ogg Theora" and Chrome supports both.

Besides video and audio, graphics are also very important in a web site nowadays. HTML5 introduces the `canvas` element. It is a drawing area that the developer can fill using the Javascript API. It supports different context renderers; at the moment, only 2d and 3d (WebGL) are supported. It can be used for games and advanced graphics.

HTML5 exposes many more features to enable developers to build advanced web sites and applications. However, HTML5 alone does not enable a developer to build rich applications. Using Javascript, developers can manipulate the DOM, build business logic rules and use several APIs the browsers provide to interact with the page and therefore engage the user. The use of cascading style sheets (CSS) enables the styling and manipulation of DOM elements, allowing a developer to customize the web page as he desires.

2.3.2 Advantages of HTML5

HTML5 is indeed bringing many new features to web browsers, including the mobile ones. The standard is not yet finished, but when it does and the most common browsers implement it completely, web applications based on it will truly be cross-platform. At the moment, most browsers already implement most parts of the standard, especially the mobile browsers of Android and iOS.

Each mobile operating system provides its own software development kit. For instance, they differ in programming languages: Java is used to build applications for Android while Objective-C must be used for developing applications for iOS. Other platforms also use other programming languages. This means that an application developed for one platform, can not be used in another without the code being ported.

This is where HTML5 has the upper hand. Using HTML5, CSS3 and Javascript, developers can target several platforms were the code once. There are already many frameworks and libraries that enable developers to use these technologies to build mobile applications, as is shown in the next section.

2.4 Frameworks for developing mobile web applications

In order to speed up the development of web applications, developers often use third-party frameworks. There are many frameworks available, the next subsections will focus on some of the most used, exploring their features, advantages and disadvantages.

2.4.1 jQuery Mobile

JQuery Mobile[2] is a framework for developing user interfaces for mobile devices based on the famous jQuery and jQuery UI libraries.

[2] http://jquerymobile.com/

JQuery Mobile provides a unified user interface system that works seamlessly across all popular mobile device platforms. It focus on being feature-rich, lightweight, with a flexible theming system and ThemeRoller tool. Including an Ajax navigation system, page transitions and a core set of UI widgets (pages, dialogs, toolbars, listviews, buttons with icons, form elements, accordions, collapsibles, and more), jQuery Mobile is one the most used frameworks for developing web apps. It is easy to learn, thanks to its simple, markup-based system to applying behavior and theming. More advanced developers can use the API of global configuration options, events, and methods to apply scripting or generate dynamic pages.

JQuery Mobile supports many devices, with the full compatibility list available in its website. "To make this broad support possible, all pages in jQuery Mobile are built on a foundation of clean, semantic HTML to ensure compatibility with pretty much any web-enabled device. In devices that interpret CSS and JavaScript, jQuery Mobile applies progressive enhancement techniques to unobtrusively transform the semantic page into a rich, interactive experience that leverages the power of jQuery and CSS. Accessibility features such as WAI-ARIA are tightly integrated throughout the framework to provide support for screen readers and other assisting technologies."[jF+12]

2.4.2 xui

Xui[3] is a micro library DOM library for building HTML5 mobile applications. It provides methods for manipulating DOM objects, handling events and effects, AJAX and styling, but, unlike jQuery, it does not provide a UI toolkit, so the developer is responsible for that.

Being a micro library and not a full framework, its second place on the rank can be explained by its inclusion in the PhoneGap framework. Xui was developed along with PhoneGap out of necessity for a framework that was mobile-oriented.

2.4.3 jQTouch

JQTouch[4] is a "Zepto/jQuery plugin for mobile web development on the iPhone, Android, iPod Touch and other devices".[Kan13a]

It is an open-source library sponsored by Sencha and has the following features: easy setup, flexible themes (using SASS), native WebKit animations, callback events, swipe detection, extensions, small file size and iOS5 scrolling.

To build an user interface the developer uses HTML and applies classes and identifiers to `div` elements which will be processed by jQTouch and transformed into pages with headers and navigation when the page loads. The interface can be themed, but is does not provide a native look&feel.

[3] http://xuijs.com/
[4] http://www.jqtouch.com/

2.4.4 Sencha Touch

Sencha Touch[5] is a high-performance HTML5 mobile application framework that enables developers to build cross-platform applications using Javascript that work on iOS, Android, BlackBerry, Kindle Fire and other devices. It features smooth scrolling and animations, more than 50 widgets, fast adaptive layout engine, native packaging, advanced lists and much more.

The designing of the user interface is done via Javascript, using the provided API. The generated UI can be themed, but it does not adapt itself to the look&feel of the platform. It is well documented, has a large community and many demo and real-world applications to learn from, making it one of the most successful frameworks for web mobile development.

2.4.5 Wink toolkit

The Wink toolkit[6] is also a Javascript framework for developing mobile web applications. It is lightweight, provides touch events handling, DOM manipulation, CSS transforms and several UI components. The interfaces are designed using Javascript to invoke their API. Available for free, it currently supports iOS, Android, BlackBerry, Bada and Windows Phone 7.

Like other frameworks described above, the generated user interface can be customized using CSS or by using one of the provided themes. And once again, it does not adapt the interface to the platform's look&feel.

It is highly modular and the existing demo applications, tutorials and community are good starting point for starting to use this toolkit.

2.4.6 Jo

Jo[7] is another framework that allows a developer to build mobile applications using Javascript and CSS. The generated UI is supposed to look native, but only on iOS. It takes full advantage of CSS3 to style the screens making it easy to customize. The documentation is good and there are a few demos, but it does not seem to be used by many applications so far.

2.4.7 Kendo UI

Figure 2.1: Kendo UI

Kendo UI Mobile[8] is a complete framework for developing cross-platform web applications that look native in Android, iOS and BlackBerry devices. Differences between platforms are

[5] http://www.sencha.com/products/touch/
[6] http://www.winktoolkit.org
[7] http://joapp.com/
[8] http://www.kendoui.com/

automatically dealt with by the framework, adapting the user interface to comply with the platform standards.

The framework promotes the separation between the application's logic and the user interface. Three central concepts define an app: application logic, which is written in Javascript, layouts and views, both written with HTML. The application logic layer manages all navigation, application history, loading views and other essential mobile app tasks, besides the logic associated with the app's business logic. Layouts define reusable portions of the application, similar to templates. Its often used to promote maintainability by defining common view areas used across multiple views. Finally, views are individual pages of the app. Most applications will have at least one view.

Currently, Kendo UI only supports iOS, Android and BlackBerry OS. Besides the native-like theming and layout system, it is also a complete mobile application framework, handling navigation, views, templates, animations and history. It can be downloaded for a 30-day trial, after which a license must be bought.

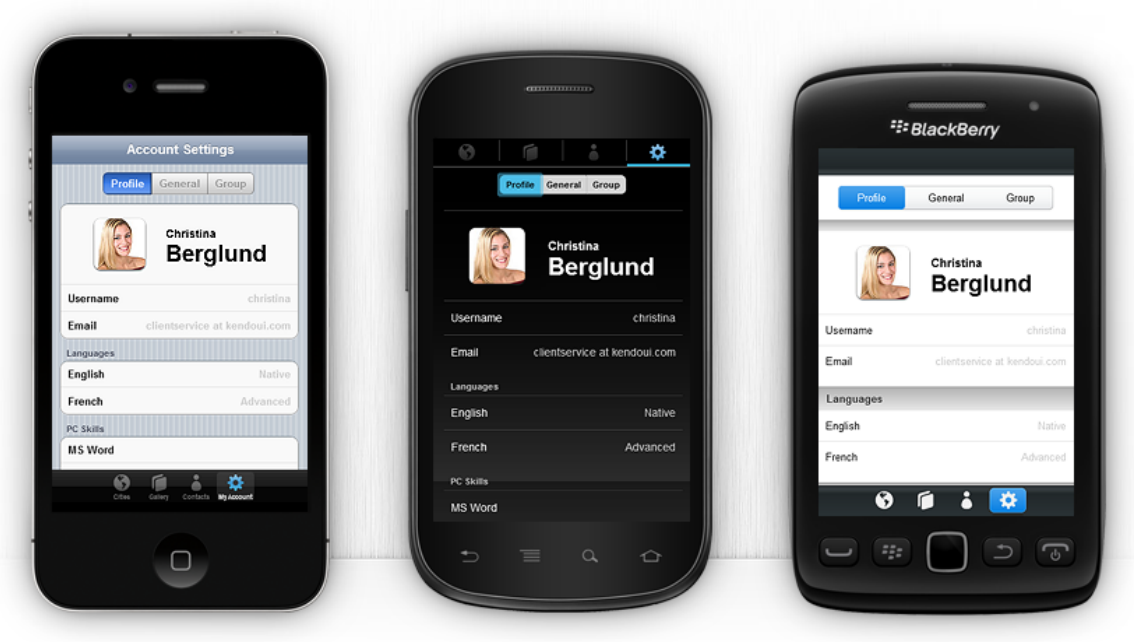


Figure 2.2: Kendo UI on iPhone, Android and BlackBerry

2.4.8 jqMobi and jqUI

jqMobi

Figure 2.3: jqMobi

JqMobi[9] is a small Javascript library for mobile HTML5 application development that can

[9] http://www.jqmobi.com/

replace jQuery in mobile devices. JqUI is a UI toolkit for building user interfaces for mobile applications using HTML5.

Using HTML5 elements such as `header`, `nav` and `footer` and some custom `data-` attributes, jqUI allows a developer to specify the user interface of a whole application and customize it through CSS and Javascript. It targets mobile WebKit browsers, so it works on iOS and Android platforms. It has a small footprint of only 22kb when gzipped and supports plugins for extending functionality.

The generated user interface can be styled using themes, but, unlike Kendo UI, it does not adapt itself to the platform's look and feel. It does provide an overall design that works nicely on mobile devices, which consists of a fixed header and footer, a content area in the middle and a menu on the side that collapses and expands on user command on small screens or is always visible in larger screens.

Through the use of plugins, the developer can extend the framework with new functionalities like sub-panels, internationalization, advertisement or maps.

JqMobi and jqUI are both open-source projects.

2.4.9 Comparison between frameworks

Despite not being a listing of all available frameworks, the list above includes enough frameworks to analyze the current state of the art.

HTML5 is still under development and the difference between platforms and their different implementations of HTML5 caused many frameworks to appear, each with its way of tackling the problem, trying to provide developers with the tools for developing better applications that run on as many devices as possible, are cost effective and user friendly.

Although all these frameworks leverage the power of HTML5, there are clearly two different approaches as to how the developer designs the applications. JQuery Mobile, jQTouch, Kendo UI and jqMobi try to extend HTML, allowing the user to write the application's screens in HTML and then applying custom identifiers, CSS classes or `data-` attributes to provide context and meaning to a specific part of the HTML so that the framework can than transform it into a part of the final application. On the other hand, Sencha Touch, Wink and Jo force the developer to design all the screens in Javascript through the use of their own API. Each approach has its advantages and disadvantages. In the first approach, the traditional HTML file and structure is maintained, the developer just has to make sure that it is structured the correct way so that the framework can interpret the file correctly. However, it is less flexible when compared with Javascript and there is the initial overhead of processing the file to prepare the user interface each time the application is loaded. In the second approach, although it is much more flexible, it requires the user to learn a whole new API. Also, the design of the screens is not as well structured as in the first approach, making it harder to be maintained by a designer that doesn't know Javascript.

As far as look&feel goes, all frameworks provide a consistent user experience across several platforms, mostly because its design and behaviour is the same on every platform. Most frameworks do support theming, but this is not enough to emulate the native look&feel of a platform.

Name	Type	HTML5-based UI	JS-based UI	Cost	Native looks
jQuery Mobile	Library	Yes	No	Free	No
jQTouch	Library	Yes	No	Free	No
Sencha Touch	Framework	No	Yes	Free/Paid	No
Wink	Framework	No	Yes	Free	No
Jo	Framework	No	Yes	Free	No
Kendo UI	Framework	Yes	No	Paid	Yes
jqMobi	Library	Yes	Yes	Free	No

Table 2.1: Comparison between frameworks

The exception to this is Kendo UI. Of all the frameworks above, it is the only one that tries to adapt the user interface to the platform's look&feel, for instance, tabs will be placed on top in Android phones, on the bottom with labels in iOS or on bottom without labels in BlackBerry. Also, the theme color and images are automatically changed to themes specific for each platform. Although it still cannot pose as a real native application, it does a better job than any other framework.

2.5 Conclusions

The smartphone world continues to evolve at an accelerated pace and developers are constantly looking for new ways to speed up the development process of their applications. Web technologies have a good support on recent mobile platforms, making them a good investment for cross-platform development, depending on the kind of application that is being developed. And combined with products like PhoneGap, web applications can pose as native applications and gain access to hardware features that otherwise they wouldn't have. This enables developers to build powerful applications using only HTML5, CSS3 and Javascript.

There are already many frameworks and libraries that aim to make it easier for the developer to build mobile web applications. However, most of these frameworks and libraries don't provide a native look&feel, rather providing themes that stay consistent across platforms. If a web application wishes to truly integrate with the platform it is running on, then the user interface should look native. However, this should be achieved without requiring the user to define multiple interfaces, one per platform.

As so, there is an opportunity and necessity in creating a tool that builds user interfaces that are automatically adjusted to the look&feel of the platform the application is running on, and it is precisely this that I propose to do in this dissertation.

Chapter 3

Case Studies

In this chapter, some known hybrid applications are analyzed to better understand the type of applications that employ web technologies and the design patterns that were used.

3.1 Facebook

The Facebook mobile application is very popular, with around 470 million mobile users, where 140 million people use the iPhone app and 176 million use the Android app.[Eva13].

Until the end of 2012, beginning of 2013, Facebook's mobile application was powered by HTML5. However, Facebook decided to change to native development, saying that betting too much on HTML5 was a mistake.[Ola12]

The developers of Sencha Touch, however, did not believe that the problem was HTML5, but rather the techniques used by the developers of Facebook, so they developed Fastbook, a proof-of-concept web application that implements the basic features of Facebook, implemented using the Sencha Touch framework. The Fastbook application even managed to beat Facebook's native Android application in terms of performance.[JA12]

In terms of user interface (see figure 3.1), both applications use a top bar with action buttons and the rest of the screen is filled with a list component, which contains the posts. A navigation drawer is also present on both applications. Swiping from the left edge or clicking on the top-left button reveals the navigation drawer, allowing the user to explore through other parts of the application.

Both the native applications (iOS, Android) and the Fastbook application use their own appearance, without adjusting to the looks of the platform, therefore looking the same on iOS, Android and even Windows Phone.[Thu13]

3.2 LinkedIn

LinkedIn is a social networking website, used mainly for professional networking, with more than 200 million members as the beginning of 2013.[Nis13]

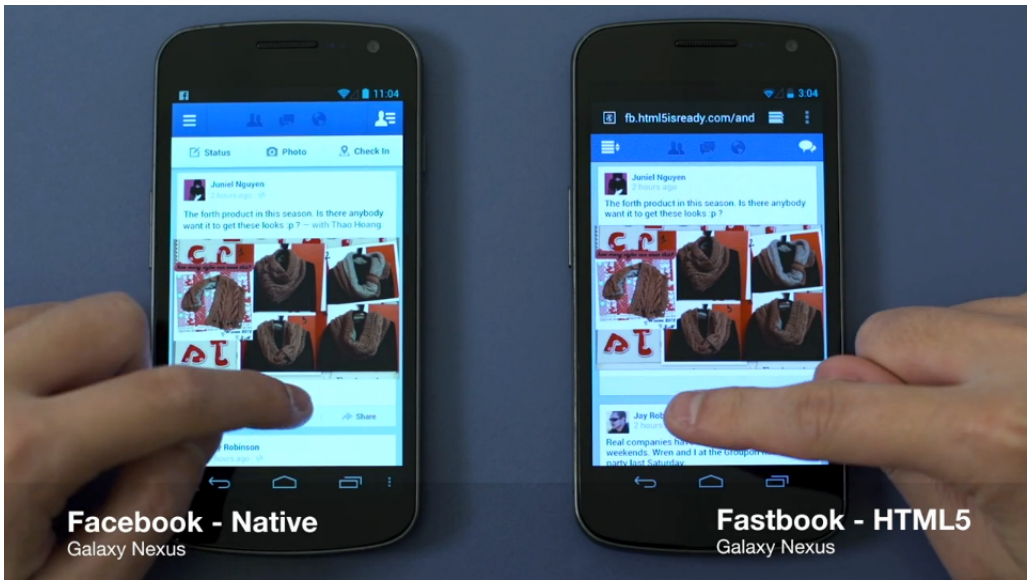


Figure 3.1: Native Android Facebook application (left) and Fastbook (right). Credit: Sencha

In mid 2011, LinkedIn launched its mobile application for Android and iOS. This application was actually a hybrid application, leveraging the power of HTML5 for 95% of the graphical interface. Later, in May 2012, they also launched an iPad application, again also powered by HTML5. Both applications were well received, for its performance and design, which was pleasant and a great improvement when compared to the web site version, which was the only interface available to mobile users before the launch of the applications. It's also worth noting that the company migrated their back-end code from Ruby on Rails to node.js, which led to an increase of performance and scalability. [O'D11, O'D12]

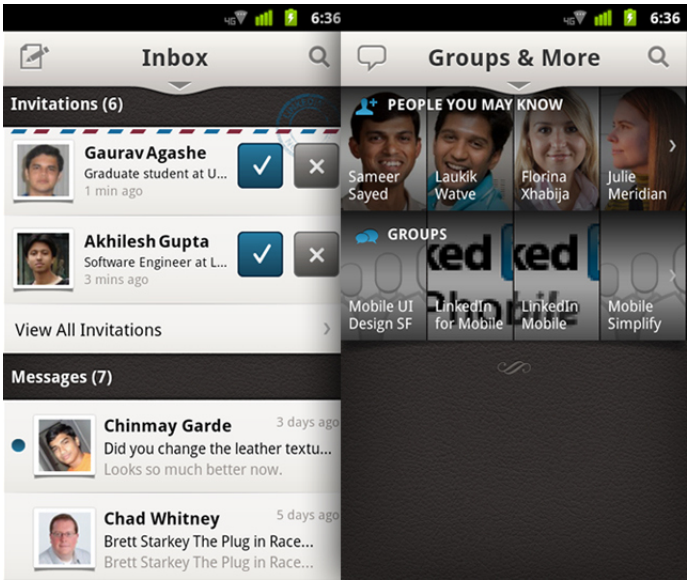


Figure 3.2: Earlier version of LinkedIn mobile. Credit: LinkedIn

Figure 3.2 shows the first mobile application that LinkedIn launched for Android and iOS. It shows a slick design, which seems like a variant of the native design of iOS, a top bar with a centered title and action buttons on each side, with a few quirks, like the arrow beneath the title of the screen indicating that it is possible to interact with the title to navigate through the application. In terms of content, lists and tiles are common across the application, similar to many other mobile applications. However, the interface of the application doesn't adapt itself to the native look of the platform, looking the same regardless of where it runs.

Later, in April 2013, LinkedIn announced new versions of their mobile applications. The novelty in these new versions is that they are completely native. LinkedIn stopped using HTML5 for building the user interface, because their users were spending more time inside of the application, leading the application to be out of memory eventually. Another major reason invoked by Kiran Prasad, LinkedIn's senior director for mobile engineering, was the HTML5 ecosystem and its lack of good developer and operations tools.[O'D13]

3.3 Exfm

Exfm is a social music discovery platform that was founded in 2010. It is powered by many music sites and, besides a web site, provides iPhone and Android applications.

Recently, the CEO of Exfm, Dan Kantor, wrote an article documenting the team's experience porting their native iPhone application to a hybrid application, using HTML5 and PhoneGap.[Kan13b] In this article, Kantor describes the key factors that led to this transformation and then presents a few technical details and their implementations.

Before changing to a hybrid application, Exfm existed in three flavors: iPhone application, Android application, web application for mobile and desktop. The Exfm team is very small, consisting of only six people, which do most of the development. However, it contracted out a lot of the native mobile application work, which has costs evidently. Discussing Exfm's future with Lucas Hrabovsky, the Chief Technical Officer (CTO) of Exfm, together they analyzed the products current state (mobile applications, Python back-end and Javascript front-end) and their hiring needs for the future. Since the six members of the team are well familiarized with Javascript, they decided to use it for everything. So, they ported the back-end to node.js and used PhoneGap for the mobile applications.

As a result of this transformation, their iOS application, which was initially rated 4 stars, went up to 4.5 stars and has more daily installs than the former native application. It also allowed the company to reduce costs by not having to hire work out. At the moment of writing, the hybrid version was only released for iPhone. The iPad and Android versions are still native applications, but the new versions are expect to be out soon.

In terms of user interface, the new iPhone application (figure 3.3) shows a different design from the Android application (figure 3.4), but also different from the native look of iOS applications. Overall, it shows a top bar with a title and action buttons on the side. The button on the left side of the title takes the user to a menu screen, allowing him to quickly navigate throughout the

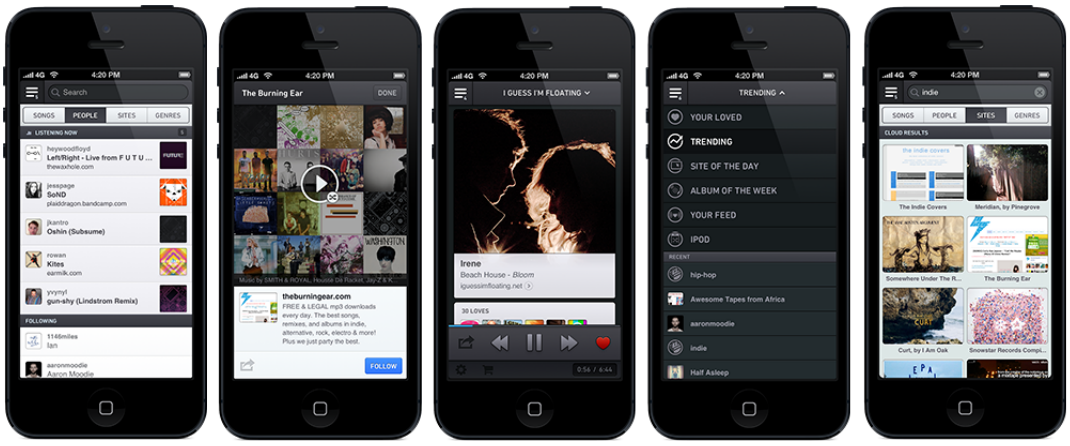


Figure 3.3: Exfm iPhone application (hybrid). Credit: Exfm

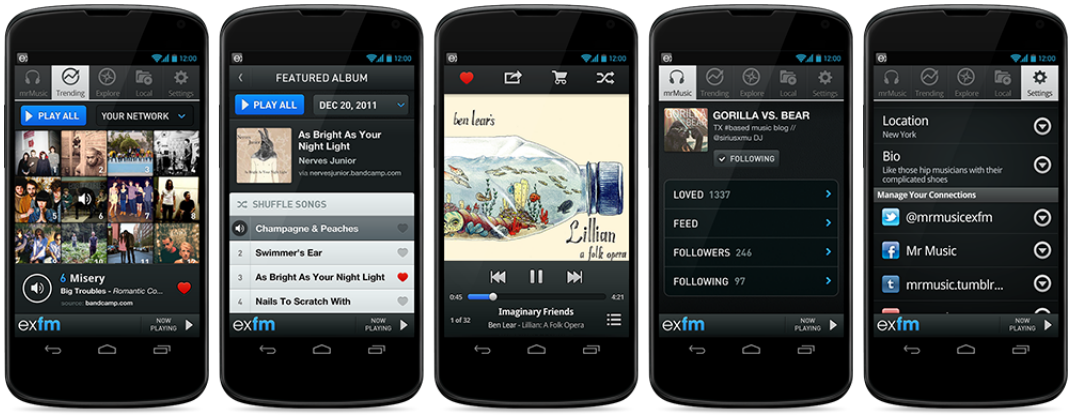


Figure 3.4: Exfm Android application (native). Credit: Exfm

application. This menu screen shows tabs that reveal lists of songs, people, sites and genres. When playing music, a footer with player controls is shown.

Since the Android is native, one can notice the use of native components, like the tab host that contains the applications primary screens and allows the user to quickly change between them, or the lists with the native arrows on the right side. Only when the Android version of the new hybrid application is released will it be possible to see if it maintains the native look of the native application or if it will look the same as the iPhone application.

3.4 Conclusion

There are many more mobile applications available, but from these three examples some conclusions can be drawn.

First, it is very interesting to see how companies and small teams bet on new technologies as HTML5. Facebook and LinkedIn are both big companies and both initially betted in HTML5 for developing their mobile applications, later migrating to native applications due to complications encountered. On the other hand, the small team behind Exfm initially betted in developing native

mobile applications, beside their main web application, and later migrated to HTML5, since they had to contract work out for developing the major part of the mobile applications. This was solution because all members of the team knew Javascript. And as result of this migration, the number of downloads increased and they managed to reduce costs. Also, the major reason for abandoning HTML5 was the lack of experience with it, memory issues and the lack of good developer tools, all factors that are expected to improve in a near future as browsers perfect their implementation of the HTML5 standard.

Another interesting and relevant fact is that these applications use basically the same look on every platform, even native ones, like Facebook's. This makes the applications stand out from an user's point of view, because the user is not familiarized with the interface components. This can be either good or bad. It can help the application distinguish itself from its competitors by promoting a different and consistent look across platforms, but it can also cause the user to look for another application that integrates better with his phone. For instance, earlier versions of Facebook for Windows Phone 8 showed a native look and feel. But when a new beta version with the same look and feel as the iOS and Android applications was released, users complained about the lack of integration with the operating system, because they are accustomed to have a native rich experience.[Thu13]

Finally, the applications share a common design pattern: top bar with title and action buttons, a menu with options that is opened when the button on the left of the title is pressed and content presented using lists and tiles. These characteristics seem to be popular with most iOS and Android applications and are a basis for the framework that was developed in this thesis.

Chapter 4

Market Study and Analysis

In order to better understand the needs of developers that develop for mobile platforms, a survey was distributed to college students and professionals.

The survey counts with a total of ninety nine (99) responses and was separated in three parts: experience; HTML5, Javascript and CSS3 and, finally, features.

The full results of the survey are available in appendix A.

4.1 Experience

The purpose on the first part, experience, was to understand the profile of the people answering the survey in terms of their experience in developing mobile applications and software in general. It comprised of five questions:

4.1.1 How many years of experience in software development do you have?

As the question states, its purpose is to collect how much experience the people answering the survey have developing software in general.

73% of the participants stated to have 1 or more years of experience, where 10%, 14%, 12% and other 10% stated to have 4, 5, 6 and 10 years of experience, respectively. We can therefore conclude that the vast majority of the participants has experience developing software in general.

4.1.2 How many years of experience do you have in developing for mobile?

This question intends to evaluate the experience of the participants in developing for mobile platforms.

38% stated to have no experience developing for mobile platforms. The rest 62% stated to have between 1 and 3 years of experience. This trend is in conformance with the rising of the importance of mobile platforms in the past years and how easy developing software for those platforms has become.

4.1.3 What platforms have you developed native applications for?

This question allows to determine the platforms the participants have developed for and, therefore, are most comfortable with. This was multiple choice question, so the sum of totals can be greater than 100%.

The platform participants have worked more with is Android, with 59% of votes, followed by Windows Phone/Mobile with 19% of the votes. The iOS platform had 8% of votes.

The fact that Android had the most votes can be explained by its vast availability in the market and the fact that the tools for developing native applications for it are free, bringing no costs in the development phase.

Although the Windows Phone platform is recent, Windows Mobile has been around since 2003, which could explain the higher number of votes when compared with the iOS platform, one of the most platforms nowadays. However, no participant stated to have more than 3 years experience developing for mobile platforms, and the Windows Mobile platform has seen its last stable update in 2010, 3 years ago, when Windows Phone first appeared. It is, therefore, impossible to determine which of the platforms the participants wanted to refer to and this is definitely an aspect to improve in future surveys.

The low vote count for the iOS platform can be explained by the high costs of developing for this platform. It requires a device capable of running MacOSX, such as a MacBook, and a developer license which costs around \$99 ($\approx$€ 77) per year.

4.1.4 What kind of mobile apps do you develop?

How the developers approach the development of their mobile applications is also important, since a hybrid approach is proposed in this dissertation.

87 participants answered this question. The majority of participants (53%) claim to develop native applications, while 32% claim to develop web applications. The rest 15% are familiar with hybrid applications.

A hybrid approach will most definitely interest the 32% that develop web applications, since they'll be able to use technologies they're used to to provide a better experience for the end user. The majority which develops native application can also be interest in an hybrid approach as a way to target multiple platforms at once, reducing development costs. The next question approaches this point of view.

4.1.5 Do you target one platform at a time, or do you have a team per platform?

The participants were confronted with three choices, regarding their approach when targeting more than one platform: one platform at a time, a team per platform or targets only one platform. Not answering this question, its assumed that the programmer targets more than one platform at a time.

Of the 65 participants that answered this question, 54% targets one platform at a time, while 40% target only one platform. 6% use one team per targeted platform.

Targeting only one platform at a time has its advantages from a managers point of view: target one platform with a small team and see if it is successful and, if it is, use the same team to port it to other platforms. And using more than one team to target several platforms is costly, although the delivery time is shortened. Through a hybrid approach, however, the application would only have to be developed once by a single team to target several platforms.

4.2 HTML5, Javascript and CSS3

The purpose of this part was to assess the familiarity and preferences of the participants towards the HTML5, Javascript and CSS3 technologies.

4.2.1 What's your skill level in Javascript?

Participants were asked to qualify their skill level in Javascript from a scale of 0 to 5, where 0 means "No skill" and 5 means "Advanced".

From the 99 participants that answered this question, 23% stated to have no knowledge of Javascript. 35% stated theirs skills to be average and above.

The adoption of the Javascript language is important for developing hybrid applications, so the fact that the votes for the levels average and above are low can be an issue. But Javascript is *de facto* programming language of the web, so there are lots of resources available for training.

4.2.2 What's your skill level in HTML5 and CSS3?

This question follows the same principle as the previous answer, but now regarding HTML5 and CSS3.

All 99 participants answered this question; 23% claimed to have no knowledge of HTML5 nor CSS3 and 44% evaluate their skills as average or better.

4.2.3 Do you prefer HTML5 or Javascript for implementing user interfaces?

There are two ways of creating a user interface using web technologies: using HTML or using Javascript. This question aims to determine the popularity of each of these approaches among the participants.

35% of the participants prefers to use HTML5 while 10% prefers to use Javascript. The majority 55% indicated to prefer to mix both technologies. Developing web applications normally involves mixing Javascript, HTML and CSS, so the results of this question are actually satisfactory.

4.3 Features

The participants were asked to rate a set of features according to their usefulness, from "Not important" to "Deal breaker".

4.3.1 Analytics

Analytics are important to know more about who uses the application and how they use it. Services like Google Analytics make this easy.

Results show that 36% of the 75 participants that answered this question consider analytics to be an useful feature. 28% consider supporting it a must, while only 3% consider it to be a deal breaker.

The developed framework will be supporting analytics indirectly through PhoneGap, since it has plugins for this.

4.3.2 Hardware acceleration

In mobile devices, hardware acceleration is very important to give the user the best experience possible, as animations and rendering in general will be smoother.

37% of the 76 participants that answered this question consider hardware acceleration to be useful, 29% consider it a must, while 8% consider it to be a deal breaker.

Through the use of CSS3 `transition` and `transform`, the developed framework will take advantage of hardware acceleration whenever it is supported.

4.3.3 Configurable effects

Smooth animations ensure a richer user experience. Using CSS3, the user can configure the animations.

Of all the participants, only 76 answered this question. 42% of the participant consider this feature to be useful, 18% consider it to be a must and 3% consider it a deal breaker.

Since the developed framework has all design stated in external stylesheets, the developer can easily extend them with new effects or configure existing ones.

4.3.4 UI Widgets

Graphical user interfaces are built using widgets, which are small components the user can interact with.

39% of the 74 that answered this question considered widgets to be useful. 27% find it to be a must and 8% considered it a deal breaker.

The developed framework provides a basic set of widgets, which will be discussed later in this document.

4.3.5 Data bindings between views and models

In the Model-View-ViewModel design pattern, it is useful to use some sort of data binding mechanism. This mechanism allows layers to detect changes in objects and automatically update themselves. What this means to the developer is that he can just update a value in the view model

and the user interface will be automatically update. Likewise, when the user interacts with the interface, the binded values are updated and the view model is notified.

41% of the 75 people that answered this question find data bindings to be useful. 31% find it to be a must, while 8% consider it a deal breaker.

The developed framework includes a data binding mechanism called “Observables”, which will be presented later on.

4.3.6 Cross-platform support

The purpose of this dissertation is to propose a framework that enables cross-platform development of applications. This question intends to clarify if developers really feel the need for cross-platform technologies.

75 people answered this question, 32% consider cross-platform support to be useful, other 32% consider it to be a necessity and 21% consider it to be a deal breaker.

Comparing to other features, this one has a high number of “deal breaker” votes, which means people do want cross-platform technologies.

4.3.7 Full SDK

A full SDK provides developers with all the necessary tools and libraries they need to build a full application.

Of the 72 people that answered this question, 35% find it to be useful while 29% and 18% find it to be a must and a deal breaker, respectively.

The developed framework will not provide a full SDK initially, but since all the necessary dependencies of the library will be compiled and minified together with the library, the developer only needs to use his favorite Javascript IDE. The demonstration application that was developed can also be used as a quick start project.

4.3.8 Documentation

A project’s documentation is very important, as it sometimes the only resource available that describe the inner works of the project and how to use it.

The responses to this question show how important documentation really is, as it was the question with more “deal breaker” responses: of the 74 participants that answered this question, 22% find it useful, 35% see it as a must, and 23

This dissertation will be used as documentation for the inner works of the developed platform. Besides this, the code will be documented and an automatic documentation tool will be used to generate the API documentation. A demonstration application was also developed in the context of this dissertation and will serve as practical example.

4.3.9 Internationalization (I18n)

Internationalization, sometimes abbreviated to i18n, refers to the ability to translate the textual content of the application to a predefined set of languages and switching between them according a setting.

37% of the 75 participants that answered this question consider support for internationalization to be useful, 31% consider it to be a must and 16% consider it to be a deal breaker.

Although the developed framework doesn't support internalization directly, Require.js, one of the core components of the framework, can be extended with a plugin[1] that allows the developer to load language bundles and use them in the application.

4.3.10 Extensions

Extensions allow the user to add a certain functionality to the framework if it is missing.

Of the 75 people that answered this question, 51% consider it to be useful, 16% and 3% consider it to a must and a deal breaker, respectively.

Since the developed framework uses Require.js, the developer can inject custom modules when necessary. The main factory method of creating widgets also allows the definition of custom widgets.

4.3.11 Monetization

Monetization is the method through which developers gain profits from their applications. One of the most common ways of monetization is through the use of advertising, embedded in the application.

Of the 71 people that answered this question, 46% consider it to be useful, 15% and 4% consider it to be a must and a deal breaker, respectively.

The developed framework has currently no standard way to show advertising or support for other monetization techniques. However, using the `Html` widget, developers can inject custom HTML, thus making it possible to inject code that shows ads.

4.4 Conclusion

The purpose of this survey was to see if the goals for this dissertation are aligned with the necessities and desires of its intended target audience. Although the number of participants was low, the responses that were obtained showed that most of the goals and proposed features where well accepted by the participants.

Most of the feedback received was taken into account when developing the framework. Decisions to drop some of the functionalities were made, mostly due to time constraints, but also because the core products that the framework can be extended through plugins to include them.

[1] http://requirejs.org/docs/api.html#i18n

This was the case for internationalization, analytics, monetization and providing a full SDK. Another decision that was taken is to support only the use of Javascript to define the user interface. Since the Javascript layer is a necessity, it was decided to leave out the support of HTML5 interfaces. However, in the future, a translator of HTML5 or any other specification language to Javascript could be developed atop of the Javascript API.

The next chapters describe the framework that was developed and the core functionalities it provides.

Chapter 5

The Meems Framework

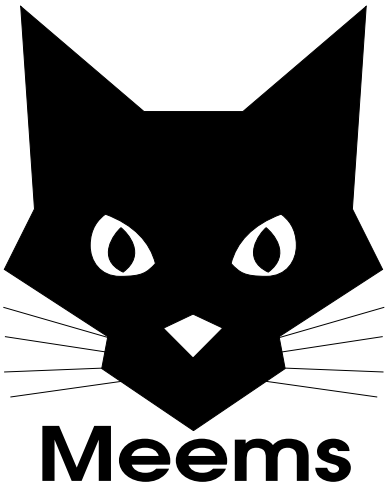


Figure 5.1: The Meems Framework logo

As stated before, most of the current mobile web development frameworks don't really address the cross-platform look&feel of user interfaces in a way that keeps both the user and the developer happy.

5.1 Goals

The Meems framework is an attempt at making the life of developers easier, while trying to provide the final user with the best user experience possible, emulating the look&feel of his mobile platform, by using the power of HTML5, CSS3 and Javascript, available in most modern mobile operating systems.

Using the Meems framework, developers are able to put together user interfaces fast, without the need to tweak each detail for each mobile platform. A single code base for all platforms, but with a native-like look&feel on each of them.

The main goal was to build a framework for developing mobile user interfaces that is completely independent, has a small footprint, a modular architecture and that is both easy to learn and to use, but also easy to tweak if necessary.

As a framework for building user interfaces, it also promotes the separation between the representation of data and the business logic, following the Model-View-ViewModel design pattern.

At the moment, together with PhoneGap, Meems allows developers to target the Android and iOS platforms, with native-like look&feel on each platform. It is possible to target other platforms, but since Meems only supports these two platforms at the moment, the developer would have to choose between one look&feel or the other.

5.2 Concepts

The Meems framework is based on a few key concepts that are essential for understanding it and how to use it.

5.2.1 Widgets

A widget is an element that influences directly the user interface, be it a control or a layout manager, it is something that users will notice and possibly interact with. Meems possesses a set of basic widgets and allows the final user to use custom widgets.

When a widget is initiated, it'll create DOM objects that will be the visual representation of the widget. These objects are usually assigned CSS classes that can be used to style the widget. The widgets are themed differently for each mobile platform supported. The widget itself can also detect the platform and behave differently in each platform. Take, for instance, the tab group widget. This widgets aggregates several named pages, showing one at once together with a navigation bar with the name of each page (called tabs). Depending on the platform, the navigation bar can either be on top of the page or on the bottom. Although configurable by the user, the default position of the bar will be different according to the platform, for instance, it will be on top on Android devices and on the bottom for iOS devices.

Widgets are also responsible for handling DOM events and mapping them to Meems internal events, converting them to meaningful events in the context of the widget. The user can then use the event handling capabilities of Meems to capture those events and process them. Taking the previous example, a tab group widget must capture the event that is triggered by the browser when the user presses a tab, change to the selected page and fire an internal event notifying the application that the tab has changed.

Widgets are the core of Meems framework. They are what allows the programmer to define the user interface in a generic way, without worrying on how it will look on each platform. If a platform is not yet supported, the widgets are the only thing that need to be updated to support it.

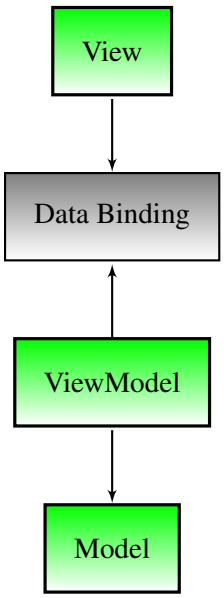


Figure 5.2: Model-View-ViewModel design pattern

5.2.2 Model-View-ViewModel

While Meems only provides the "View" part of an application, its API is oriented towards the separation of data representation and the application logic according with the Model-View-ViewModel (MVVM) design pattern.

The MVVM design pattern promotes the separation of the application's logic from the user interface. The transfer of data between the two is done through data bindings and commands. Three base concepts are introduced:

- **Model**, a domain object, representing the data that is to be user or manipulated somehow. It is free of business and representation logic.
- **View**, the user interface where the data is presented and/or collected. The view can contain behaviours, events and data-bindings and does not maintain state.
- **ViewModel**, acts as a translator between the model and the view layers, performing all the necessary translations of the data. It exposes methods, commands, helpers to maintain the state of the view and events. It can manipulate the view in response to changes in the model and update the model when the view changes.

The MVVM pattern was first mentioned to the public by Microsoft's John Gossman in a blog post titled "Introduction to Model/View/ViewModel pattern for building WPF apps". As the title suggests, this pattern is an adaptation of the Model-View-Controller design pattern for Windows Presentation Foundation (then codenamed Avalon). In his post, Glossman explains that the purpose of this design pattern is to allow designers and programmers to collaborate, making the View a sole responsibility of the designer, instead of the developer having to translate mock-ups

and designs to code. This scenario fitted Microsoft's new markup language for interface design, called XAML, that can be generated by tools designed to simplify the job of designers, most notable Microsoft's Expression. Using declarative data-bindings, the developer could bind the application's data to the user interface the designer built. So, data-binding is another base concept for the MVVM pattern.

Data-binding is a mechanism that allows the View to notify the ViewModel whenever the user changes the data and also allows the View to be notified when the Model has suffered changes, normally triggered by an action of the user. This is particularly useful in complex user interfaces, specially when the same data is used in several components.

Meems also aims to help both the developer and the designer. However, since it is in an early stage, the specification of the user interface is done using Javascript. At later stages of development, and probably outside of the context of this thesis, a mechanism of translation from HTML5, or some other markup language, to Javascript will be developed.

After the user interface is built, the developer can bind data objects to the controls through the use of observables.

5.2.3 Observables

Observables are data containers enhanced with a mechanism for notifying all the interested parties when its contents have been modified.

Following the Observer design pattern, each observable stores a list of interested observers, that will be notified whenever the content of the observable changes. Observables possess methods for subscribing and unsubscribing observers at runtime.

Observables are the main mechanism for exchanging data between the user interface (View) and view models in the Meems framework. Most properties of the available widgets support observables and will be automatically updated when a change occurs. Likewise, whenever the user changes the data through the user interface, the view model will be notified of the change so that it can update itself.

Observables are used for listening for changes in specific properties, instead of entire objects. The example code below shows how to use observables in the Meems framework.

```
1  // Defining the model, each property must be an observable.
2  var person = {
3      name : Meems.Observable.observable("John Smith"),
4      age  : Meems.Observable.observable(18),
5
6      // Arrays must be declared with observableArray.
7      children : Meems.Observable.observableArray([])
8  };
9
10 // Register an observer.
11 var onNameChanged = function (oldValue, newValue) {
```

```
12      console.log("name changed from " + oldValue + " to " + newValue + "!");
13  });
14
15  person.name.subscribe(onNameChanged);
16
17  /* To change the value of an observable, invoke it as a
18     function with the new value as argument. */
19  person.name("John Adams Smith");
20  person.age(20);
21
22  /* To retrieve its current value, invoke as a function
23     with no arguments. */
24  console.log("My name is " + person.name() + " and I'm " +
25              person.age());
26
27  // Remove the observer.
28  person.name.unsubscribe(onNameChanged);
```

Listing 5.1: Observables in Meems

5.3 Architecture

In order to maintain the code clean and modular, the Meems framework is separated in four libraries:

- **meems-events**, a library for manipulating DOM and custom events, providing an easy API and cross-platform support;
- **meems-utils**, a library with multiple utilities, divided by domain. Includes methods for manipulating arrays, maps, functions and others;
- **meems-scroll**, emulates fixed `div` scrolling, since older devices don't allow `div`'s to scroll when they are positioned absolutely;
- **meems-ui**, the core library that contains the UI widgets and methods to build the user interface.

The first two modules are independent and can be used separately. Meems-scroll depends on meems-events and meems-utils and meems-ui depends on the other three. Besides these dependencies, there is only one more, which all of them rely on: `require.js`[1].

`Require.js` is a Javascript file and module loader that implements the Asynchronous Module Definition proposal[2]. It enables the separation of the code in modules. A module can depend

[1] http://requirejs.org/
[2] https://github.com/amdjs/amdjs-api/wiki/AMD

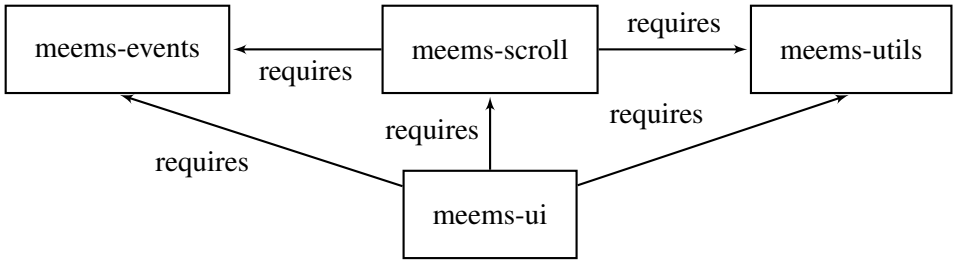


Figure 5.3: Dependencies between the libraries that constitute the Meems framework

on other modules. Given a main module, `require.js` will dynamically load it and all its dependencies. This enables the programmer to keep the code modular, easier to maintain and to debug. Another advantage of having the code modular and using `require.js` is the usage of its optimizer tool. This optimizer tool, called `r.js`, can run on top of Node.js or Rhino and will take as input Javascript modules and produce a single Javascript file, minified to the smallest size possible. This will reduce the number of files that the client must download and also its total size. This is a very powerful step when preparing the code for production. It can also optimize CSS, but the Meems framework does not use this, yet.

5.3.1 AMD - Asynchronous Module Definition

The Asynchronous Module Definition API, also known as AMD, "specifies a mechanism for defining modules such that the module and its dependencies can be asynchronously loaded. This is particularly well suited for the browser environment where synchronous loading of modules incurs performance, usability, debugging, and cross-domain access problems."[ao13]

Using the AMD API to describe a module, dependencies are declared using strings, therefore reducing the use of global variables. The API also allows to map a module name to different paths, which makes it easy to swapping implementations. This can be useful for creating mock objects for testing purposes.

A module and its dependencies are declared using the `define` method. The first argument is usually an array of strings containing the dependencies of the module. These dependencies are then passed as arguments to the module constructor, which is a function passed as second argument.

```
1 define(function () {
2     return {
3         sayHello: function () {
4             // Do something.
5         }
6     };
7 });
```

Listing 5.2: Defining a module without dependencies in AMD (modA.js)

```
define(function () {
    function ModB() {
        return this;
    }

    ModB.prototype.doSomething = function () {
        // Do something.
    };

    return ModB;
});
```

Listing 5.3: Defining a module without dependencies in AMD (modB.js)

```
define(["modA", "modB"], function (ModA, ModB) {
    return {
        init: function () {
            ModA.sayHello();
            var b = new ModB();
            b.doSomething();
        }
    };
});
```

Listing 5.4: Defining modules in AMD (modC.js)

5.3.2 Meems-events

Event processing is a must when programming for the Web. The DOM triggers events in response to the actions of the user (for instance, when he moves the mouse, presses a key, touches the screen, scrolls the page, etc.) and also to notify the pages of some internal events, such as timers or when loading ends.

Applications can bind listeners to events in order to be notified when such event is triggered. This is how the application can react to the user actions. However, registering this listeners is not standardized among all browsers. There are two major event registration models: W3C's and Microsoft's. The first model is supported by Mozilla, Safari, Chrome, Konqueror and other WebKit-based browsers. The second model is supported by Internet Explorer. Opera supports both models. They both offer the two basic functions of event registration (add and remove a listener), but with a different syntax. W3C's model provides the `addEventListener` function for registering listeners and `removeEventListener` to remove them.Microsoft's model provides the `attachEvent` function for registration and the `detachEvent` for removal of listeners. Other differences between this models do exist, but being outside of the scope of this dissertation,

the reader is encouraged to read other material about this topic. A good starting point is the QuirksMode website: http://www.quirksmode.org/js/events_advanced.html.

Meems is designed for mobile environments, which nowadays is almost as diverse as the desktop environment in terms of Web rendering. Android, iOS and BlackBerry and others are Webkit-based, Windows Phone uses Trident and Firefox OS uses Gecko. This fragmentation is the reason why the meems-events library was made. It is a layer of abstraction over the browser's event model, providing the programmer with a simple API for manipulating event listeners, that is supported on most platforms and that can be easily extended should the need arise.

Packed under the `Dom` package name, the meems-events library exposes two core functions for manipulating DOM events:

- **`on`**`(element, event, listener)`: Register a new `listener` of `element` for the given `event`;
- **`off`**`(element, event, listener)`: Remove the `listener` waiting for `event`s on `element`.

Besides abstracting from the browser model, the meems-events library also provides an API to implement custom events that can be triggered and listened for. For this purpose, the `Handler` class is provided. This class should be extended by any class whose objects will trigger custom events. It exposes the following methods:

- **`on`**`(event, listener)`: Register a new `listener` for the given `event`;
- **`off`**`(event, listener)`: Remove the `listener` waiting for `event`s;
- **`fire`**`(event, args)`: Triggers the given `event`, invoking all registered listeners, passing them the provided `args`.

5.3.3 Meems-utils

Like any other programming language, Javascript does not provide the user with every possible functionality, rather it provides the building blocks, that programmers can use to build libraries to solve specific problems. These libraries can then be used by other projects to reduce implementation time.

Although Meems could use external libraries in order to speed up development, the need for a small, targeted code that runs well fast on mobile environments was a critical factor in deciding on building meems-utils instead on using libraries such as jQuery or Zepto. The whole Meems framework is smaller that jQuery (both minified, 57 kB versus 93 kB, respectively)!

Meems-utils is a library that provides several helper methods in a few domains, separated by packages: `Ajax`, `Fn`, `Array`, `Map` and `Dom`.

5.3.3.1 Ajax

This package exposes the `request` method which enables the application to build an AJAX request and receive the server's answer easily. It's specially useful when developing a layer for communicating with services that follow the REST pattern. This method works across all platforms that support the `XMLHttpRequest` object. It supports custom headers, setting the method (GET, POST, DELETE, PUT, HEAD) and request and response formatting (JSON, XML, URL encoded).

5.3.3.2 Array

This packages contains functions to manipulate and extract information from arrays.

The `indexOfByProp` function accepts an array, a property name and a value to match as arguments and returns the position of the first element of the array whose requested property matches the given value.

The `remove` method accepts an array and an element as arguments and proceeds to remove from the array (in-place) the first appearance of the element.

The `moveElement` method swaps the position of two elements inside an array.

5.3.3.3 Dom

To allow applications to interact with pages, browsers expose to Javascript the DOM tree of pages. DOM, which stands for Document Object Model, are objects that represent elements in the page, have methods and properties that define how the browser renders them and are organized in a tree hierarchy.

Manipulating the DOM tree of a page can, however, be costly. When the DOM tree is changed or the properties of an elements are changed, the browser must calculate the changes to the user interface and paint it accordingly. Depending on the changes, this process can lead to the painting of large areas of the screen, which is very bad, performance-wise.[Hen12]

To ensure that an application runs as smoothly as possible, the interactions with the DOM must be keep to a minimum and changes should be batched to resize the number of reflows as must as possible. It is, therefore, useful to have helper methods that store all changes and apply them in bulk when requested.

One of the possible approaches to apply bulk changes to an object is to use CSS classes. Adding and removing classes from the object will cause the browser to update the object with the properties defined by the CSS classes and to update the screen if needed. To keep the repaints to a minimum, the classes associated with object should be communicated to the DOM once. This is what the `Dom` package does.

This package provides the following functions:

- `setClass`, queues a request to replace the class names associated with the given object with a new one;

- `addClass`, queues a request to add a new class name to the given object;
- `removeClass`, queues a request to remove a class name from the given object;
- `getClass`, returns a string with all the class names associated with the given object, including the ones that weren't yet applied;
- `setHtml`, queues a request to change the `innerHTML` property of an object;
- `getHtml`, returns the contents of the `innerHTML` of the given object, including changes that weren't yet applied;
- `applyChanges`, calculates all changes that are queued and applies them to the objects.

For example, assuming an object `obj` and the following code:

```
1  // On module 1 (replaces all classes with the ui-button class)
2  obj.className = "ui-button";
3  ....
4
5  // On module 2 (adds ui-button-normal class)
6  obj.className += " ui-button-normal";
7  ....
8
9  // On module 3 (removes ui-button class name)
10 obj.className = obj.className.replace(/\\bui-button\\b/, "");
11 ....
```

The final classes for `obj` would be `ui-button-normal`. Instead of modifying the DOM directly 3 times, it could be replaced with the following code:

```
1  var Dom = Meems.Utils.Dom;
2
3  // On module 1 (replaces all classes with the ui-button class)
4  Dom.setClass(obj, "ui-button");
5  ....
6
7  // On module 2 (adds ui-button-normal class)
8  Dom.addClass(obj, "ui-button-normal");
9  ....
10
11 // On module 3 (removes ui-button class name)
12 Dom.removeClass(obj, "ui-button");
13 ....
14
15 // At the end of all processing:
16 Dom.applyChanges();
```

This will cause a single update to `obj`'s `className` property, with the advantage of making the code look cleaner and simpler.

5.3.3.4 Fn

Javascript relies heavily on functions and callbacks so it is very practical to have some helper methods. This package provides those helpers.

Being a prototype-based programming language, the concept of class is not a base concept of the language as in pure object-oriented languages. Javascript makes use of functions that, together with the `new` operator, create new objects based on a prototype, therefore simulating the behaviour of object-oriented programming.

The `extend` method is syntactic sugar for implementing class inheritance. This method is automatically installed in Javascript's `Function` object, so it's accessible through every function ("class"). Using this method, the inheritance chain is maintained, meaning that the `instanceof` operator works just fine and can be used to verify if an object is an instance of a certain class or one of its sub-classes.

```
1  function Animal() {
2      return this;
3  }
4
5  Animal.prototype.name = function () {
6      return "animal";
7  };
8
9  function Cat() {
10     return this;
11 }
12
13 Cat.extend(Animal, {
14     "name": function () {
15         return "cat";
16     },
17
18     "meow": function () {
19         return "meow";
20     }
21 });
22
23 var meems = new Cat();
24 var isMeemsACat = meems instanceof Cat; // true
25 var isMeemsAnAnimal = meems instanceof Animal; // true
```

Listing 5.5: Using the `extend` helper

The `postPone` function postpones the execution of a function to end of the current processing stack. This is useful for running commands only after the current processing is done and give the browser the opportunity to do other things, like rendering. It is syntactic sugar for the `setTimeout` method.

The function `bind` returns a new function that invokes the function given as the first argument with the object passed in the second argument as the `this` object. Useful for when the `this` object is different depending on the context and the developer needs it to be a specific object.

`Throttle` allows the developer to limit the invocation of a function over time. Using this method, a new function is created that, when invoked, will invoke the wrapped function at most once in the specified time interval. This is useful is situation when a callback can be invoked many time very quickly, causing the CPU to spike and performance to degrade. An example of this is a window resize event handler. When the user resizes the browser window, the handler will be invoked several times until the resizing finishes, so that the application can resize itself to the browser area. Wrapping the handler in a `throttle` call with a small interval (around 100ms), it's possible to avoid CPU spikes while still updating the application's user interface.

5.3.3.5 Map

This is a small package, with only one function: `getKeys`. This functions returns an array with the keys of the object that is passed as argument.

5.3.4 Meems-scroll

The Meems framework intends to simulate native user interfaces using HTML5 and Javascript. A very common scenario in native applications is the existence of a fixed header and/or footer, which provide the user with actions or information.

Although HTML element property `position` can have the value `fixed`, which dictates that the element will stay in the same position regardless of the scrolling of the page, this value is currently not very well supported by mobile platforms.[3] Known behaviours include totally ignoring the property, let the element scroll with the page only returning it to its position when the scrolling motion finishes and flashing when scrolling.

To overcome the lack of support of `position:fixed` elements, one must use absolute positioning of elements. But with this, another problem arises. When the contents of an absolutely positioned element exceed the element's dimensions, the user should be allowed to scroll to view the rest of the contents when the `overflow` CSS property is set to `auto` or `scroll`. However, this is not supported on some mobile platforms, like some versions of Android[4]. Recent versions of iOS and Android do support setting `-webkit-overflow-scrolling: touch;` on elements to enable native scrolling, but it does not allow customization and on some platforms, like iOS<5, requires the user to use two fingers to scroll the content.[5] To overcome this behavior and

[3] http://caniuse.com/#feat=css\discretionary{-}{}{}fixed
[4] http://code.google.com/p/android/issues/detail?id=2911
[5] http://goo.gl/2lAfy

ensure a consistent scroll experience, it's necessary to use Javascript to simulate the scrolling of elements. The meems-scroll module emulates the scrolling behaviour of Android and iOS on absolute positioned elements, using hardware acceleration when possible to minimize the impact on the application's performance. It uses CSS3's `transition` and `transform` properties to animate the elements, allowing the browser to use hardware acceleration to render smooth animations.

It is very simple to use. The module returns a class, internally named `Scroll`, which accepts the element which must be scrollable and an object with configuration settings as arguments in its constructor. The following configuration settings are supported:

Setting	**Description**	**Default**
axisLock	After starting to scroll, lock the scrolling to the axis of the movement or allow it to change while scrolling?	true
bouncing	Should the contents bounce if scrolled past the limit or remain static at the limits?	true
disableTouchEvents	Scroll only programmatically?	false
fadeOutDuration	How long should the fade out animation of the scrollbar take (in seconds)?	1
friction	Friction factor for scrolling. The higher the slower the scrolling is.	100.0
hideScroller	Should the scrollbar be hidden?	false
minDistanceOfDrag	When starting to scroll, how many pixels must the user's finger move for the element start to scroll?	10
paging	Use the size of the element as page size and only allow the user to scroll one page at a time?	false
snap	Snap the scrolling to the given page size (in pixels). 0 to disable snapping.	0
scrollX	Allow the content to scroll horizontally?	false
scrollY	Allow the content to scroll vertically?	true
timingFunction	CSS3 timing function when animating the contents after the scrolling finished.	ease-out
totalMaxTime	The maximum time in seconds the animation of the contents can take.	1

Table 5.2: Available settings in meems-scroll

```
1 // Add meems-scroll as a dependency
2 define(["meems-scroll"], function (Scroll) {
3     var scroller;
4     return {
5         init: function () {
```

```
6             // Obtain the element that will be scrolled.
7             var elm = document.getElementById("elmToScroll");
8             // Create a scroller.
9             scroller = new Scroll(elm, {
10                paging: false,
11                snap: 0,
12                scrollX: false,
13                scrollY: true
14            });
15        },
16
17        destroy: function () {
18            if (scroller) {
19                /* Destroy the scroller when
20                   it's no longer needed. */
21                scroller.destroy();
22                scroller = null;
23            }
24        }
25    };
26 });
```

Listing 5.6: Using meems-scroll

5.3.4.1 Implementation

When associating a scroller with an object, meems-scroll calculates the dimensions of the container (the element which contents will scroll) and the dimensions of the contents inside the container. This information is stored in the container's DOM element as custom properties. Then, additional DOM elements representing the scrollbars are attached to the container, one for the horizontal axis and another for the vertical axis, depending of the scroller's configuration. Scroller will manage this elements, showing and hiding them when needed and destroying them when their no longer necessary. Also, events handlers are attached to the element so that meems-scroll can process the events that are triggered when the user presses the container, drags it and releases it and simulate the scrolling of the contents. To know which events to listen for, meems-scroll will first detect if the `touchstart` event is supported. If it is, this means the platform supports touch events (probably it's a mobile device with a touch screen), so it will listen for the `touchstart`, `touchmove` and `touchend` events. If touch events aren't supported, it'll listen for normal mouse events: `mousedown`, `mousemove` and `mouseup`.

When the user starts pressing the container to initiate a scrolling motion, the scroller catches this event and performs the following actions: obtains and stores the position of the cursor or touch point, calculates the size of the contents of the container, updates a flag indicating that the container is scrolling and prevents the event from being processed by the browser. This last step

is very important to overcome a defect[6] in the Android platform (2.0, 2.1).

When the user moves the finger or the cursor while the container is in scrolling motion, the cursor position is grabbed and subtracted from its initial position, stored when the user started pressing the container. The result of this difference will dictate how many pixels the content is moved and in which direction. The position of the content is updated by using the `transform` property together with the `translate3d` value. This enables the browser to use hardware acceleration when rendering the content, providing a smoother scrolling experience. The scrollbars are also updated to reflect the current position of the content relative to the container: the size of the bar will be equal to $\frac{min(containerSize,contentSize)}{contentSize} \times containerSize$ and its position will be equal to $\frac{-contentPosition}{contentSize} \times containerSize$. These will be calculated for each of the axis, being each dimension chosen according to the axis: size will be the `height` for the Y axis and the `width` for the X axis, position will be `top` for the Y axis and `left` for the X axis.

When the dragging motion is released, the content will be animated, continuing to scroll for a small period of time at a velocity that depends on the movement performed by the user when dragging the content. To perform the animation, the CSS3 property `transition` is used. To use it it is necessary to know the final position of the content and how long should it take until it reaches the final position. To calculate this, meems-scroll uses as input the duration of the dragging motion ($time$) and how many pixels the content was moved ($offset$). From this, the average velocity in pixels per second is calculated ($velocity = \frac{offset}{time}$). A first estimate of the animations total time is then calculated by taking the absolute value of the average velocity and dividing it by a constant value ($totalTime = abs(\frac{velocity}{friction})$). Having an estimate for the time, an estimate for the final position is calculated: $finalPos = currentPos - velocity \times totalTime$.

The estimate of the final position must be adjusted according to the constraints imposed by the scroller's configuration, for instance, paging, snapping and the minimum and maximum possible value.

If paging is enabled, the final position will be adjusted to
$round(\frac{currentPos \pm containerSize}{containerSize}) \times containerSize$, depending on the direction of the movement.

If snapping is enabled, the final position will be adjusted to $round(\frac{finalPos}{config.snap}) \times config.snap$.

Then, the final position is adjusted according the its limit values. It will always be zero or a negative value. If the size of the content is smaller than the size of the container, the final position will always be zero, since there is nothing to show beyond the container's boundary. If the content surpasses the container in size, then the maximum absolute value for the position of the content is $contentSize - containerSize$. The final position is truncated, if necessary, to be inside this boundaries.

After the final position is calculated, the animation's duration is adjusted as well by taking into account the difference between the final position before and after being adjusted: $totalTime = min(totalTime \times abs(\frac{startPos - newFinalPos}{startPos - finalPos}), config.totalMaxTime)$.

This calculations are performed for each axis and the highest time will be used as the duration of the animation.

[6] http://code.google.com/p/android/issues/detail?id=5491

Finally, the content properties `transition` and `transform` are updated:

```
content.style.transition = "all " + totalTime + "s " + config.timingFunction;
content.style.transform = "translate3d(" + finalPosX + "px," + finalPosY +
"px,0)";
```

During the animation of the content, the position of the scrollbars still needs to be updated. To do this, an asynchronous cycle of duration equal to that of the animation of content is performed using the `requestAnimationFrame` method. In each iteration of this cycle, the current position of the content is retrieved using the `getComputedStyle` function and the scrollbars' positions are updated accordingly.

5.3.5 Meems-ui

The meems-ui module is the main module of the Meems framework. It contains the widgets that will be the building blocks developers use to construct the user interface of a mobile application.
It follows an object-oriented approach, based around the `Widget` class. Every widget must extend this class and override at least the `update` method. Widgets must also be registered in the `Meems.UI` factory class, through the `registerWidget(name, class)` method. After this, to instantiate a widget, the `create` factory method of the `Meems.UI` namespace can be used. It takes the name of the type of widget to create as argument and returns an instance of that type of widget.
The base widgets provided with the framework can be separated in three categories: navigation and organization, layout and user interface controls.

5.3.5.1 Navigation and organization

The widgets in this category are used to specify the high-level structure of the application's user interface. Mobile applications are usually composed of multiple screens which transition from one to another.
In the context of the Meems framework, an application can have one or more `Page`s, where only one is visible at a time. To store and transition between pages a `PageHolder` is used.
One common user interface pattern in mobile applications nowadays is a lateral menu that slides from the side. This allows the user to easily access other options of the application without leaving the current screen. In Meems, this is called an `Aside`.

PageHolder is a widget that can store multiple pages and transition between them. It should be the top-level widget in an application. Pages can be set using the `pages` function and transitions can be accomplished by invoking the `currentPage` function with the desired page as the first argument. It imposes no restrictions regarding what pages can transition to others, it's up to the

application itself to decide this. Transitions between pages can be animated using CSS3. A default transition animation is provided in the `effects.css` file.

Page represents a screen of the application. It exposes three facets: header, content and footer, as this structure is the most common in mobile applications. Although each facet can contain any widget, it is best to use the `Header` and `Footer` widgets for the header and footer, respectively. For the content facet, usually a group or list widget is used, the most common would be the `List`, `Form` and `TabGroup` widgets.

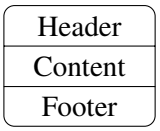


Figure 5.4: Page structure

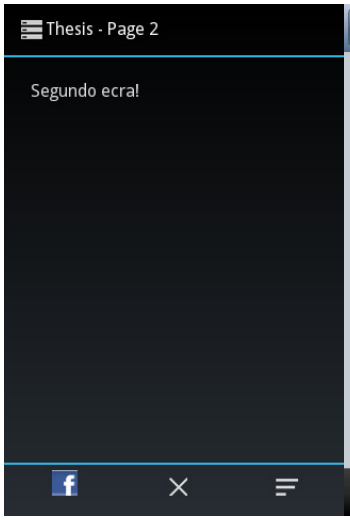


Figure 5.5: Page in Android

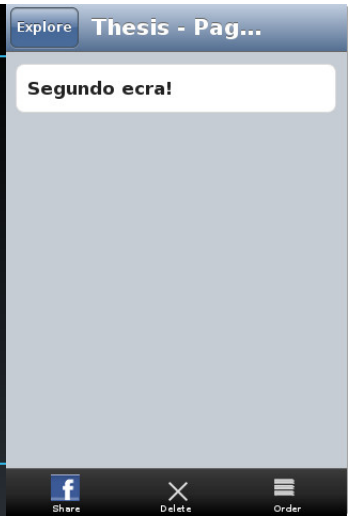


Figure 5.6: Page in iOS

Aside enables the use of one page as a lateral menu that can be toggled when on a small screen and that will be always visible on large screens. The purpose of this is to provide the user with easy access to all the major pages of the application.

5.3.5.2 Layout

Some widgets are meant to organize other widgets, placing them in the correct positions with the correct dimensions. The layout of a screen can be achieved by using these widgets.

Button Group is a widget that organizes buttons in an horizontal manner. Using the `maxButtons` attribute, its possible to limit the number of buttons visible at a time. If more buttons than those allowed exist, an overflow button will be made visible. Clicking this button pops up a menu with items representing the extra buttons.

Footer is a widget meant to be used with the `Page` widget. This widget adds a footer to the screen, where a button group can be placed with the `buttons` facet.

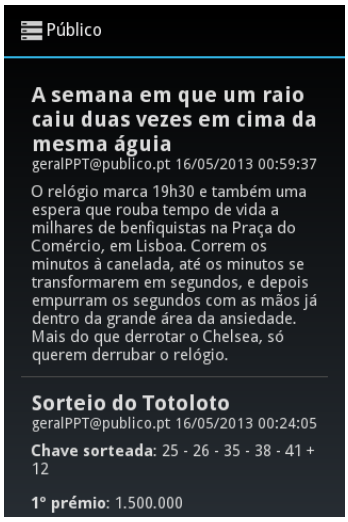


Figure 5.7: Menu closed

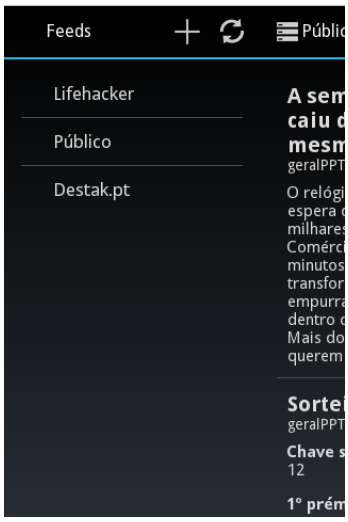


Figure 5.8: Menu open

Figure 5.9: Menu always visible

Figure 5.10: A button group in Android (left) and iOS (right)

Form stores fields that accept user input and organizes them vertically. A form can also have a title, which will be placed above the first field.

Group is a widget that merely groups other widgets vertically.

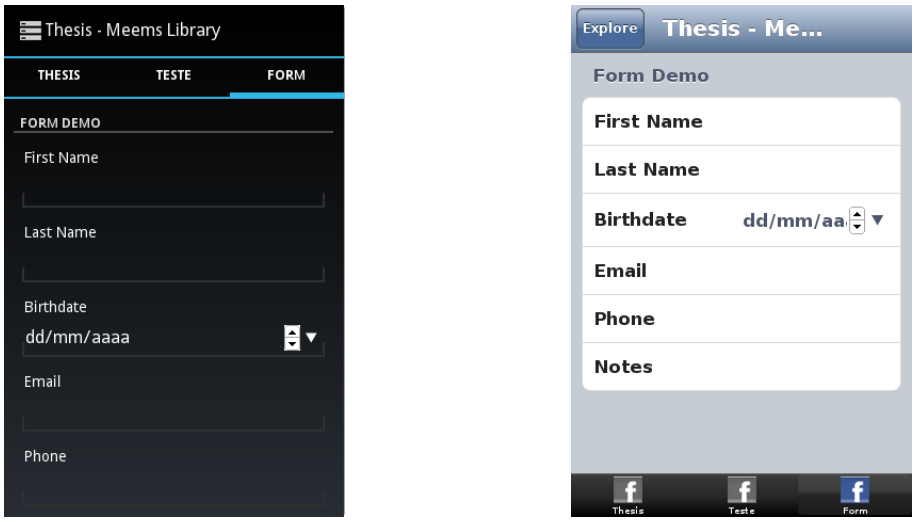


Figure 5.11: A form example in Android (left) and iOS (right)

Header like the `Footer` widget, is meant to be used with the `Page` widget. It adds a header bar to the screen, which contains a title and two facets, `buttonsleft` and `buttonsright`, to place buttons in left and right side of the title, respectively. These facets expect `ButtonGroup` widgets. The title can be set through the `title` attribute.

List is a representation of an array. Each item of the array will be converted to a graphical representation using a Mustache template, that can be set using the `template` attribute. Header items are also supported. An element will be converted to a header if its `header` property is true. Header elements are converted using the template defined by the `headerTemplate` attribute. Besides representing a list of items, this widget also provides mechanisms that allow the user to manipulate the items, namely, sort them manually and selecting several items before an operation through the use of checkboxes. To enable sorting, the `sortable` attribute must be set to true. To enable multiple selection, the `selectionMode` attribute must be set to `multiple`.

Tab Group allows to separate content in categories, through several partial pages, called `Tabs`. Only one tab is visible at a time and each one is associated with a named button that allows navigating between tabs. The `tabs` method can be used to associate an array of `Tabs`.

Tab is a widget meant to be used with the `Tab Group` widget. The contents of the tab can be set through the `content` facet. The `title` attribute is used to set the text that will appear in the tab button.

5.3.5.3 UI controls

The following widgets are basic units usually used to present a single piece of information or collect information from the user.

Figure 5.12: A simple list (left) and a list with ordering and multiple selection (right)

Button allows the user to trigger a single action. Is characterized with a text and an icon, set through the `title` and `icon` attributes, respectively. The context in which the button is used will d:etermine the visibility of the button's text and icon. The platform will also influence this. For instance, in the Android platform, buttons inside a `Header` will appear as an icon only, while on the iOS platform they'll show only as text. See figure 5.10 for an example of buttons in a header and figure 5.12 for an example of a button inside a form.

Slider allows a user to select a value from a fixed range by sliding its finger/cursor over a bar. The widget exposes two attributes to manipulate the range of valid values: `minimum` and `maximum`. To read or write the value of the slider, the `value` method must be used. The value can either be a numeric value or an observable.

Switch is usually used to represent or collect a boolean value. Like the `Slider` widget, its value can be read or written using the `value` method, which also supports observables, besides boolean values.

Text Field allows to capture textual information from the user. The `value` method can be used to read or write the contents of the field. A string or an observable can be used. A text field also exposes the `type` attribute, which allows to specify the type of data that is meant to be collected according with the HTML5 standard for input controls. This includes types like: text, number, date, tel (for phone numbers), email and search. Setting the appropriate type allows the platform to adjust how it collects the data. For instance, if the type is date, a date picker will be presented to the user instead of a keyboard. If it is email, the keyboard layout will be adjusted to place the symbol in a more handy place.

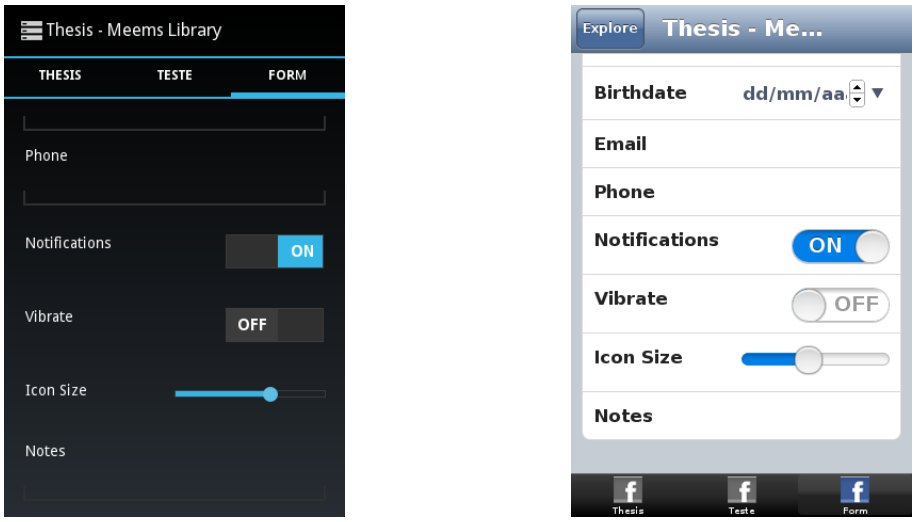


Figure 5.13: Example of switches, sliders and text fields in Android (left) and iOS (right)

Html is widget that allows the developer to use Mustache templates for generating HTML elements, making it ideal to present information to the user. Mustache[7] is a logic-less template language that uses double braces to denote a placeholder. The placeholders of the template are then replaced with values corresponding to the values of the properties of an object named after the placeholders. For example, given the template `This is a {{expName}}.` and the object `{"expName":"test"}`, the final result would be `This is a test.`. The template can be set using the `html` attribute. The object to be bound to the template can be set using the `data` attribute.

5.3.6 Building user interfaces

The Meems Framework is meant be used as a presentation layer for mobile applications. Following the Single Page Application (SPA) pattern, all the screens of the application are present in a single page and made visible or hidden to transition between screens, therefore avoiding the need to refresh the page to present new information, providing the user with a more fluid experience and emulating the behaviour of a native application.
Since all the application's screens are present in the same page, it's necessary to use a mechanism to manage those pages. The `PageHolder` widget is responsible for storing all pages and transition between them upon command. It must, therefore, be the root widget of an application. Pages can then be added to this widget.
To create all the necessary widgets, the factory method `create` must be used. Passing it a widget type, it'll return an instance of that type that can be immediately modified using its chainable methods. Meems provides this chainable API so that it is possible to chain several method invocations, reducing the amount of code that needs to be written.

[7] http://mustache.github.io/

After creating all the necessary widgets, the HTML DOM elements that will represent the widgets must be created and added to the page. The creation of the elements is achieved by invoking the `update` method of the root widget. This will trigger the `update` method of itself and all its children, ensuring that all the widgets associated with it have DOM elements ready to be added to the DOM tree. To give the developer more flexibility, the developer must manually insert the root DOM element as child of the element he sees fit as parent. Most of the times, it will be the body of the page (`document.body`). Invoking the `el` method of the root widget returns the DOM element that must be added to the page. See the example below for how to create a simple user interface and how to insert it in the DOM tree.

```
1  // Create a page with a simple list.
2  var page1 =
3      UI.create("page")
4          .facet("header", UI.create("header").attr("title", "Page 1"))
5          .facet("content",
6              UI.create("list")
7                  .items([
8                      { text: "Item 1" },
9                      { text: "Item 2" },
10                     { text: "Item 3" },
11                     { text: "Item 4" }
12                 ]));
13
14 // Create a second page with another simple list.
15 var page2 =
16     UI.create("page")
17         .facet("header", UI.create("header").attr("title", "Page 2"))
18         .facet("content",
19             UI.create("list")
20                 .items([
21                     { text: "Item 5" },
22                     { text: "Item 6" },
23                     { text: "Item 7" },
24                     { text: "Item 8" }
25                 ]));
26
27 // Create the root widget and add the pages.
28 // By default, the visible page is the first one.
29 var root = UI.create("pageholder")
30              .pages([ page1, page2 ]);
31
32 // Create all elements, add them to the page and apply all changes.
33 root.update();
34 document.body.appendChild(root.el());
35 Utils.Dom.applyChanges();
```

Listing 5.7: Creating a simple user interface

Besides defining the interface in Javascript, it is also necessary to add the CSS files corresponding to the current platform. These can added dynamically through Javascript or can be added manually when there is a main HTML file per platform. Each platform has three CSS files that must be imported:

- **ui.css** - Contains all the CSS necessary to render the widgets correctly;
- **effects.css** - Contains CSS for animating widgets;
- **icons.css** - Contains icon graphics in Base64 format.

```
1  function loadCss(urls) {
2      "use strict";
3
4      var link,
5          head = document.getElementsByTagName("head")[0],
6          firstSibling = head.childNodes[0];
7
8      for (var i = 0, ln = urls.length; i < ln; ++i) {
9          link = document.createElement("link");
10         link.type = "text/css";
11         link.rel = "stylesheet";
12         link.href = urls[i];
13         head.insertBefore(link, firstSibling);
14     }
15 }
16
17 var platform = Utils.Dom.userAgent();
18 loadCss([
19     "css/meems/" + platform + "/ui.css",
20     "css/meems/" + platform + "/icons.css",
21     "css/meems/" + platform + "/effects.css"
22 ]);
```

Listing 5.8: Dynamically insert CSS files

However, the creation of the user interface is not enough to create a mobile application. When the user interacts with the interface, the logic layer must be invoked and update the interface properly. To detect the interactions of the user with the interface, events are triggered. The logic layer can listen for those events and respond appropriately.
To update the information on the user interface and also passing information to the logic layer, it is recommended to use observables. Changing the value of the observables will cause the widgets associated with the observable to be updated. Likewise, the logic layer can attach observers to observables and react whenever their values change.

5.3.7 Effects

To animate the widgets when their state changes, Meems uses CSS3 `transition` and `animation` properties. Using these properties enables the host browser to use hardware acceleration to provide smoother animations.
For changing the state of the widgets of the application, Meems changes class names. So, using CSS3, it's possible to detect these changes and start the appropriate animations. For instance, to show a widget, the class name `ui-hide` is removed and `ui-show` is added. So, combining these class names and CSS3 animations, it is possible to, for instance, animate a page transition, as shown in the following snippet:

```
1  /* Define a new animation */
2  @keyframes fadeInRight {
3      0\% {
4          opacity: 0;
5          transform: translateX(20px);
6      }
7      100\% {
8          opacity: 1;
9          transform: translateX(0);
10     }
11 }
12
13 /* Set the parameters of the animation. */
14 .ui-page {
15     -webkit-animation-fill-mode: both;
16     -moz-animation-fill-mode: both;
17     -ms-animation-fill-mode: both;
18     -o-animation-fill-mode: both;
19     animation-fill-mode: both;
20     -webkit-animation-duration: 0.25s;
21     -moz-animation-duration: 0.25s;
22     -ms-animation-duration: 0.25s;
23     -o-animation-duration: 0.25s;
24     animation-duration: 0.25s;
25 }
26
27 /* When a page is made visible, execute fadeInRight animation. */
28 .ui-page.ui-show {
29     -webkit-animation-name: fadeInRight;
30     -moz-animation-name: fadeInRight;
31     -o-animation-name: fadeInRight;
32     animation-name: fadeInRight;
33 }
```

Listing 5.9: Animate page

At the moment, it is necessary to use the vendor prefixed version of the `animation` properties, since most browsers still don't recognize the normalized version.
The animations can also be platform dependent, so there are placed in a `effects.css` file inside the folder of each platform's theme.

5.3.8 Icons

Graphical items take an important role in nowadays applications, especially icons, since they help the user to quickly identify the type of action a certain button performs or the importance of some information.
Icons are usually stored and distributed as independent image files, most commonly in the JPEG, PNG or GIF format. In the Meems Framework, however, base icons are distributed in a single CSS file, `icons.css`, that contains all the images the application needs. The file contains at least one CSS class per image, where the `background-image` property value is the image itself encoded as a data URI. A data URI is a file encoded following the URI schema. Its format goes as follows:
`data:[<mime type>][;charset=<charset>][;base64],<encoded data>`

```
1 .ui-icon-loading ,.ui-icon-loading.ui-disabled {
2     background-image: url('...<truncated>...');
3     background-repeat: no-repeat;
4 }
```

Listing 5.10: An icon represented in CSS as a data URI

This approach has some advantages and disadvantages.
Placing all icons inside a CSS file will make the file have a considerable size, but it will also reduce the amount of HTTP requests the browser must make to retrieve all icons. So, if the application has 50 icons the browser would have to make 50 HTTP requests to the server, but if their are all inside a single CSS file, only one request must be made. The overhead of opening new connections to the server is therefore replaced by taking longer to download a single file, which most of the times compensates. Besides, enable GZip in the server helps reduce the size of the file when the transfer takes place and telling the browser that it can cache the file, it would only have to be downloaded every once and a while. This way, the application can load faster.
Another advantage of this approach is the possibility to use CSS media queries to recognize the device's resolution and use the appropriate version of the icons. With the advent of high resolution displays such as the Retina display, images must be of a higher resolution too in order to look crisp and nice. There are two possible approaches to solving this problem: have all the versions of the icons inside a single CSS file and use media queries inside of the CSS file, or have one version of the icons per file and only include the relevant one. The first approach would produce a very large file that would be hard to maintain and every device would download the same file, independently of its resolution. The second approach allows for clean separation of versions and reduces the amount of data the application must download. Consider the following two files: `icons.css`,

which contains 32x32 pixel version of all icons, and `icons2x.css`, which contains the 64x64 version of the icons. Using the following HTML tags in the head of the main HTML document would be enough to ensure that the correct version of the icons is applied depending on the device resolution:

```
<!-- Medium icons -->
<link rel="stylesheet" href="css/theme/android/icons.css "/>
<!-- Retina icons -->
<link rel="stylesheet" href="css/theme/android/icons2x.css" media="(-webkit-min
    -device-pixel-ratio: 2), (min-resolution: 192dpi)" />
```

Listing 5.11: Use different icon resolutions according to device

The problem with this is that both files will still be downloaded. To overcome this, use Javascript to detect the device's pixel ratio and include only the appropriate style sheet:

```
var platform = Utils.Dom.userAgent(),
    pixelRatio = window.devicePixelRatio,
    suffix = pixelRatio >= 2 ? "2x" : "";
loadCss([
    "css/meems/" + platform + "/ui" + suffix + ".css",
    "css/meems/" + platform + "/icons" + suffix + ".css",
    "css/meems/" + platform + "/effects.css"
]);
```

Listing 5.12: Using Javascript to detect the pixel ratio and import the appropriate CSS files

A problem with keeping all the icons in one CSS file it's maintainability. It is much easier to edit, save and publish a single image file than maintaining a huge single text file that contains all the icons in an application. Also, a new step must be introduced in the process of adding or editing icons of the application: the conversion of the image file to data URI. There are a lot of online converters that can perform this, for example, http://dataurl.net/#dataurlmaker. However, the advantages stated above largely pay off the extra effort that must be put when creating the icon themes.

5.4 Documentation

This dissertation contains explanation about the basic concepts of the framework and its inner works, so it can be used as an initial reading document. Besides this, all the developed source code is documented using Javadoc-style annotations, that are later processed by the YUIDoc[8] tool, that is used to generate HTML files containing the API documentation.

[8] http://yui.github.io/yuidoc/

5.5 Conclusions

The differences between browsers present in the most common mobile platforms and desktop browsers require developers to take precautions and to throughly test their applications in real devices. Workarounds for defects present in older versions of the platforms are also needed if one wishes to support them. This was especially true for the support of `overflow: scroll`. Due to the lack of native support from the browsers, a Javascript solution, meems-scroll, had to be developed and integrated into the framework, resulting in extra work to accomplish something that the browser was supposed to provide.

The similarities between the iOS and Android platforms in terms of design made the implementation easier. Adding support for new platforms such as Windows Phone or BlackBerry OS 10 will pose a great challenge, since the graphical guidelines are a bit different.

Finally, developing big applications and libraries in Javascript demands good practices. The flexibility of the language is an advantage, but can also be a curse if not controlled. The use of Require.js helped maintaining the code modular and clean, separating each unit of the code in its file and allowing to declare their dependencies, so that the developer doesn't have to worry about the order in which to import the files. It was also helpful for compressing and minifying the code in order to reduce load times and facilitate distribution.

Chapter 6

RSS Reader: Demo Application

To demonstrate how to use the Meems Framework and its behaviour in a real-world application, a small RSS news reader application was developed.
RSS, which stands for "Rich Site Summary", is web feed format used to publish frequently updated works in a standardized format. An RSS document, commonly called feed or channel, can include many items, or news. These information about each item can include full or partial text content of the item plus meta-data like the publishing date and author.
Using an RSS reader, like the one describe in this chapter, people can subscribe to the feeds of their favorite sites and consult all their updates in a single software, instead of having to check each site individually.

6.1 Requirements

Since this application is only for demonstration purposes, there are only a few base requirements. The application must allow the user to authenticate in order to have access to his personal feed directory. When authenticated, the user can consult a list of his feed subscriptions, press a subscription to views its news and select a news to consult it. The user must also be able to manage its subscriptions: add a new subscription by URL, remove subscriptions and order them manually.

6.1.1 Functional Requirements

6.1.1.1 Authentication

The user must authenticate using his email and password. The application will communicate with the back-end server to validate the credentials. The server will check if the user exists in the MongoDB database and if its hashed password matches the hash in the database.
If the authentication fails, the user must be warned about it and stopped from proceeding to the next screen.
If the authentication finishes successfully, the initially empty news screen is shown and the list of feeds the user has subscribed is loaded.

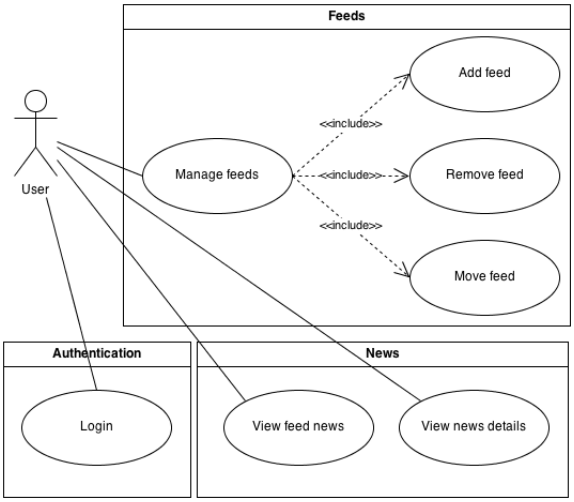


Figure 6.1: Simplified use case diagram

6.1.1.2 Feeds

The application must load from the back-end server the list of feeds the user has subscribed. This list will be presented to the user, so that he can choose which feed to open.
Upon clicking on a feed, the application will load from cache all the news of the feed. If the news in cache are too old (around 30 minutes), the application will ask the back-end server for an updated list of news and store them in cache.
The user will also have the possibility to manage feeds. This includes adding a new feed, removing existing feeds and manually ordering the feeds. All changes can then be saved and sent to the back-end server for storage, or can be discarded by returning to the previous screen without saving. When adding a new feed, the user must provide the URL, which will be sent to the back-end server. The server then tries to read the feed and obtain its name, stores it in the user's account and returns the name of the feed to the application so that it can show it to the user.

6.1.1.3 News

After the user selects a feed, its news are listed. Each news has a title, an author, a publishing date and a small description. These should be visible in the main listing. The user will be able to scroll through the list to view more news.
When the user clicks on a news article, its full contents will be shown in a separate screen, by loading the page of the original website that provided the feed.

6.1.2 Non-functional Requirements

The project must be separated in two major parts: a mobile application and a back-end server. This separation allows for future improvement and scalability.
The mobile application must run on Android (2.3+) and iOS (5+) platforms, with a look and feel as closest to native applications as possible.

All user information, which includes email, password and feed subscriptions, must be stored in a database.

6.2 User Interface

Figure 6.2: Low fidelity mockup. From top-left to bottom right: login screen, empty news screen, feeds screen, manage feeds screen, news screen, news detail screen

The user interface of the mobile applications must look and behave like a native application. Since the target platforms are Android and iOS, the same basic UI design patterns apply. The applications will have a header with button actions and a title. In iOS, buttons will be used, while in Android an action bar will be used instead. A footer will also be present when managing the feeds. Lists will be used for presenting the list of feeds and news and a form will be used for the login screen and the subscription of a new feed.

The most important design pattern used is the navigation drawer. The user will be able to click a "Feeds" button which reveals a side pane, the drawer. This drawer will contain the list of feeds,

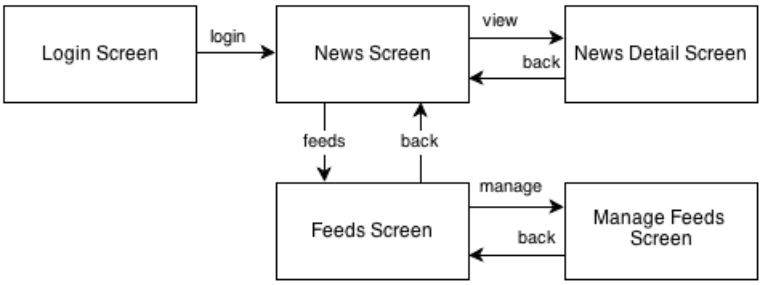


Figure 6.3: Flow between the screens

that allow the user to navigate through his subscriptions, and also more actions related with the feeds, such as the "Manage" action.

6.2.1 Login Screen

The login screen will have two input text fields, one for the email and another for the password. The user must fill in both fields and press the "Login" button.
If the authentication fails, a pop-up alert will warn the user about the error. After correcting his credentials, the user can try again.
If the authentication succeeds, the news screen will be shown.

6.2.2 News Screen

This screen will show the list of news, which is initially empty until the user selects a feed to read. To choose a feed, the user has the button "Feeds" on the top of the screen, which, following the navigation drawer design pattern, will reveal a side pane with the list of feeds. If the resolution of the device is big enough (width larger that 650 pixels), this side pane will always be visible. Clicking on a feed will close the drawer, update the page title and show the list of news of that feed. When the user clicks on a news, the news detail screen is shown.
Besides showing the list of feeds, the navigation drawer also exposes an action relative to the feeds: the "Manage" action. Clicking on this button, the manage feeds screen will be shown in the side pane.

6.2.3 Manage Feeds Screen

This screen enables the user to subscribe to new feeds, by entering (probably pasting) the URL of the feed and pressing the "Add" button. The URL will be appended to the bottom of the list of feeds.
The list of feeds will allow the user to select multiple feeds by pressing the checkboxes on the left of each item. When one or more feeds are selected, a footer will appear with the option to delete the selected feeds.

On the right side of each item, an order button is present. Pressing and dragging this button, will cause the item to be moved in list according to the movements of the user. Releasing the button will effectively place the item in its new place in the list.
All these operations are applied to a copy of the user's feed list. If the user presses the "Back" button on the top-left corner of the page, the copy is discarded and all modifications are lost. However, if the user clicks on the "Save" button, the modifications are sent to the back-end server and are applied locally, after a successful reply from the server. After saving, the user be taken back to be previous screen.

6.2.4 News Detail Screen

This screen will allow the user to view all the details of a news article, by embedding directly (through an iframe) the website the news originates from. The link for the article necessary for this operation is provided by the back-end server, which provides it together with the list of news. When the user is done, clicking on the "Back" button will bring him back to the news screen.

6.3 Architecture

The project is divided in two parts: a mobile application that uses the Meems Framework and require.js and a back-end server implemented using Node.js as the runtime, the Express framework and MongoDB as the storage engine.

6.3.1 Back-end Server

Node.js[1] is a server-side Javascript platform, based on Google Chrome's Javascript runtime, that uses an event-driven non-blocking I/O model, providing a lightweight environment for scalable real-time applications. And since it runs Javascript, it was possible to implement the server and the client using a single programming language.
The simplify and accelerate the building process of the back-end server, the Express framework was used. This framework purpose is to facilitate the development of web applications using node.js, by providing a simple API to deal with request routing, middleware and response building.
For storing the information of all users, MongoDB is used. MongoDB is a powerful document database and is the leading NoSQL database.[2] The paradigm behind MongoDB is different from that of relational database management systems (RDMS). Instead of schemas and tables, MongoDB has collections and documents. Collections have a name and store several documents. No schema or structure is imposed to documents, which are normally represented using the JSON notation. Since JSON, which stands for "JavaScript Object Notation", can easily be parsed in Javascript and both the server and the client are implemented in Javascript, it is easy to load and use the data directly from the database.

[1] http://nodejs.org/
[2] http://www.10gen.com/leading-nosql-database

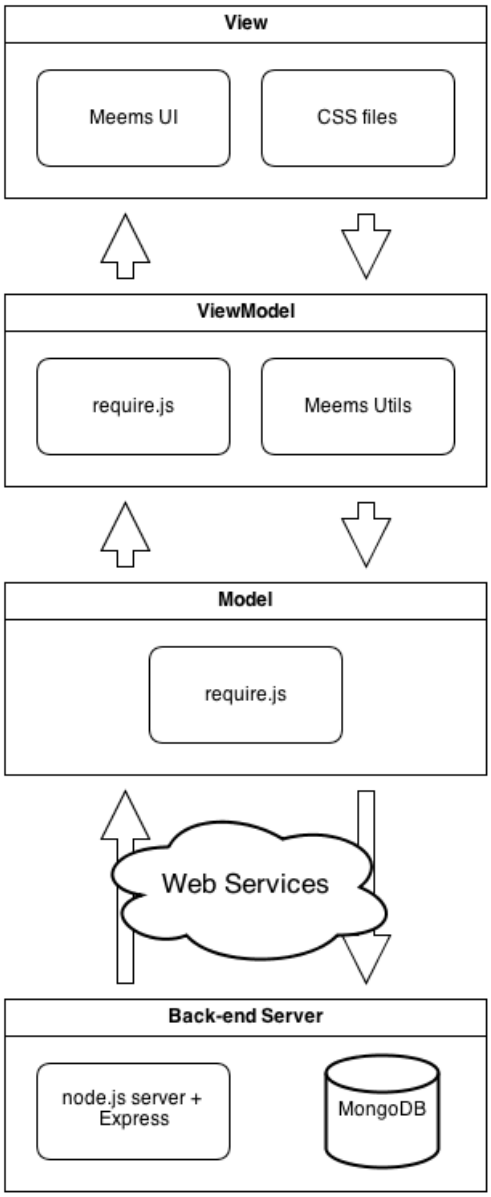


Figure 6.4: Architecture overview

6.3.1.1 Data model

The information related to each user is divided in two collections: `users` and `feeds`.
The `users` collection contains a document per user. The documents contain the following fields:

- **email** - The email of the user, is used as his identifier and must be unique.
- **password** - A SHA1 hash of the user's password. Is used for authenticating the user.

The `feeds` collection contains all the feeds that the users have subscribed, one per feed. Each document stores the following fields:

- **owner** - The email of the user that owns the feed.

- **name** - The title of the feed.
- **url** - The URL of the feed.

6.3.1.2 Web Services

To expose to the mobile application the functionality provided by the server, a REST-based API is made available, exposing two web services: `users` and `feeds`.
The API goes as follows:

Method	Path	Description
`GET`	`/user`	Get user account information of the currently logged user
`POST`	`/user`	Create a new user account. Expects JSON object with the `email`, `password` and `name` of the user
`PUT`	`/user`	Update a user account. Expects JSON object with the `email`, `password` and `name` of the user
`POST`	`/user/login`	Login a user. Expects JSON object with the `email`, `password`
`GET`	`/feeds`	Returns all the feed subscriptions of the logged user
`GET`	`/feeds/:id`	Returns the latest news of the feed subscription identified by `:id`
`POST`	`/feeds`	Logged user subscribes to a new feed. Expects JSON object with the `url` of the feed.
`PUT`	`/feeds`	Updates feed subscriptions in bulk. Expects a JSON array with feed objects inside.
`PUT`	`/feeds/:id`	Update a single feed subscription, identified by `:id`. Expects a JSON object with `name`, `url` and `order` (for changing the order).
`DELETE`	`/feeds`	Removes feed subscriptions in bulk. Expects a JSON array of feed id's.
`DELETE`	`/feeds/:id`	Removes a single feed subscription, identified by `:id`

Table 6.1: Back-end server API overview

Each entry point is protected and requires the user to first authenticate using the `/user/login` service. This service returns a session id in the format of a cookie that must be present in all future requests. Otherwise, the server will refuse to complete the request with a `403 Forbidden` response.

6.3.2 Mobile Application

To develop the mobile application, the Meems Framework was used for the View layer and require.js and pure Javascript together with a few helper methods from the meems-utils package were used for the ViewModel and Model layers.

In order to simplify the development phase, models are simple Javascript objects, so these are not represented directly by a file or module. The downside of this simplification is that the communication with the web services has to be made from the ViewModel layer. While this is not the standard way, this application is for demonstration purposes only.
Require.js was used to separate all code units into modules, one per file, and to automatically manage dependencies. The application code is structured in the following way:

- **css** Contains all the CSS files, including Meems themes
- **js** Contains all the Javascript files
  - **lib** External libraries: Meems and require.js
  - **view** Contains the Javascript files that build the views
    - **feeds.js** Implements the screen "Feeds"
    - **login.js** Implements the screen "Login"
    - **news.js** Implements the screen "News"
    - **newsdetail.js** Implements the screen "News Detail"
    - **phone.js** Implements the overall application view, acts as glue for the other views and is the main view that instantiates and controls all others
  - **viewmodel** Contains the Javascript files with the application logic. These files implement the ViewModels that form the bridge between the views and the web services.
    - **feeds.js** Implements the code for managing feeds
    - **login.js** Implements the code for authenticating the user
    - **news.js** Implements the code for listing and presenting news of a feed
    - **newsdetail.js** Implements the code to view a news detail information
    - **phone.js** Implements the code that initialize other ViewModels and manages navigation between screens
  - **index.js** Main module of the project, it's the file imported by require.js. Contains all the initialization code: imports the necessary CSS files according to the platform, initializes the main ViewModel and install the main view's root element
- **index.html** The application's entry point. Imports require.js, telling it where to locate the main module, index.js

The view modules use Meems UI to build each page and then expose it to the view model by exporting a Javascript object with the `ui` property set to the root widget of the page, usually a `Page` or `Aside` widget. When creating the view, the module will also instantiate all the necessary observables, bind them to the widgets and expose them through the same object as the `ui`, so that the view model can update the view in response to some event. It is also common for the view to capture low-level events from Meems UI and translate them to higher-level events. For instance,

capturing a `button:pressed` event triggered by a `ButtonGroup`, detecting which button was pressed, for example the "Manage" button, and triggering a `feeds:manage` event that can be captured and processed by the view model.
The view model modules have its corresponding view module as a dependency, so Require.js will inject the view modules at runtime. All modules have an `init` method that initializes all that is necessary. Views create the interface and view models call the views' `init` method, install event handlers and update observables to their initial value.

6.3.2.1 Login View

The login view is the page where the user must authenticate himself. The root widget is a `Page` with a `Header`, which contains a submit button, and a `Form` as content. The form contains two input fields: email and password. When the user presses the submit button, a `login` event is triggered. The view model then captures this event and attempts to login.

6.3.2.2 Feeds View

This view builds the page that shows the list of feeds the user has subscribed to and also the page that allows the user to manage them. This view will be presented as a side pane, so the root widget is a `PageHolder` that holds two pages: feeds and manage feeds.
The "Feeds" page is where the feeds the user subscribed are listed, so that he can choose one to see its news. The root widget is a `Page` widget, with a `Header` and a `List` for content. In the header, there is a button to enter the "Manage Feeds" screen.
The "Manage Feeds" view allows the user to manage its feeds. The root widget is a `Page` with a `Header`, which contains a back and a save button, and, as content, a `Form` and a `List`. The form is for subscribing to a new feed. It has a URL input field and an "Add" button. The list is multiple selection, sortable list with the feeds of the user. Whenever more than one feed is selected, a `Footer` with a delete button is shown. If the user presses the back button, all changes are lost and the user is taken back to the "Feeds View". The user must therefore click the save button in the header for the changes to apply.

6.3.2.3 News View

The "News" view is where all the news of the currently selected feed are presented. The root widget is a `Page` with a `Header` and a `List` as content. The header has a "Feeds" button that toggles the side pane which contains the "Feeds" view. The list contains all the news of a feed and when one of its items is selected by the user, the `news:clicked` event is triggered, so that the view model can show its details. To show all the necessary details of each news item, the item template of the list had to adjusted, as shown below.

```
1 var itemTemplate =
2     "<div class=\"news\">" +
```

```
        "<div class=\"title\">{{title}}</div>" +
        "<div class=\"date\">{{formattedDate}}</div>" +
        "<div class=\"author\">{{author}}</div>" +
        "<div class=\"summary\">{{&summary}}</div>" +
        "<div class=\"clear\"></div>" +
    "</div>";

var list = UI.create("list")
    .items(news) // Bind to an observable array.
    .template(itemTemplate) // Change the item template.
    .attr('empty', '<b>No news.</b><br>' + // Change the empty message.
                   '<br>Please select a feed from the menu.')
    .attr('style', 'normal')
    .attr('sortable', false)
    .on("item:clicked", function (eventName, item) {
        pageNews.fire("news:clicked", item);
        return true;
    })
```

Listing 6.1: Changing a list's item template

6.3.2.4 News Detail View

The "News Detail" view appears after the user selects a news article. The view model will obtain the original link for that news and show this view. The root widget is a `Page`, with a `Header`, which contains a back button, and a `Html` for content. The header's title will be the news' title. The html widget has a template and a data object binded to it. The template is an `iframe` whose source is the `link` property of the data object. So, when the news item is binded to the html widget as the data object, it's link will be used as the source of the iframe. The user will then be able to view the original news article.

```
var pageTemplate = "<iframe class=\"news-details\" src=\"{{link}}\"></iframe>";
var content = UI.create("html")
    .attr("customClass", "ui-fill") // Makes the iframe fill its parent
    .attr("html", pageTemplate) // Change the template
    .attr("data", news) // Bind the news data object (an observable)
```

Listing 6.2: Creating an iframe using an `Html` widget

6.3.2.5 Phone View

This is the main view of the application. Its root widget is an `Aside`, which will be used for placing the "Feeds" view as a side pane. As the main content, a `PageHolder` that contains the "News" and "News Detail" views is used.

6.3.2.6 Login View Model

The login view model initializes the "Login" view and deals with communicating with the back-end server when the user presses the submit button.
When initializing, the view model checks if it has the credentials of the user stored in the `localStorage` and, if it has, it attempts to login automatically using those credentials.
Otherwise, the user will have to fill in his email and password and press the submit button. When the user presses the submit button, the `login` event is captured and the view model collects the email and password the user wrote using the observables binded to those fields and submits an AJAX request to the back-end server, to the `/user/login` endpoint.
If the user is successfully authenticated, the credentials are stored for automatic authentication the next time the user opens the application and the "Feeds" view is shown.
However, if the authentication fails, the user is warned by ways of an alert and remains in the same page so that he can try again.

6.3.2.7 Feeds View Model

This view model initializes the "Feeds" view and handles the events generated by it.
The `feeds:clicked` event is triggered when the user clicks on a feed. The view model will check its cache (which is stored in `localStorage`) and collects the list of news from there. If the cache is outdated, it'll ask the back-end server for new news. After collecting the array of news to show, it passes them to the "News" view model, invoking its `showNews` method. This method updates the observables that set the title and the list content of the "News" view. After that, it'll show the "News"
When the user presses the manage button in "Feeds" view, the event `feeds:manage` is triggered and captured by this view model. It creates a copy of the current feeds of the user, that will be used as target of all the user's modifications when on the manage screen, and makes a transition to the manage screen.
The `feeds:add` event is triggered when the user is in the "Manage feeds" view, inserts an URL and presses the "Add" button. When this happens, the view model reads the value of the observable associated with the URL field and adds a new Javascript object to the temporary feeds array. This object contains the properties `name` and `url`, both with the new URL.
The `feeds:cancel` event is triggered when the user presses the back button. It causes the view model to clear the temporary feeds list and return to the "Feeds" screen.
The `feeds:save` event if triggered when the user presses the save button. The view model captures this event and merges the temporary list and the present list, identifying which feeds to add, edit or delete. Then, for each of these operations, it'll communicate the changes to the back-end server using the API described before.

6.3.2.8 News View Model

This view model initializes the "News" view. It implements the `showNews` method mentioned before and also an event handler for the `news:clicked` event, that is triggered when the user selects a news article. When this event is triggered, the view model passes the select news article to the news detail view model and makes a transition to the news detail view.

6.3.2.9 News Detail View Model

This view model initializes the "News Detail" view and updates it every time the user selects a different news article to read.

6.3.3 Deployment and Packaging

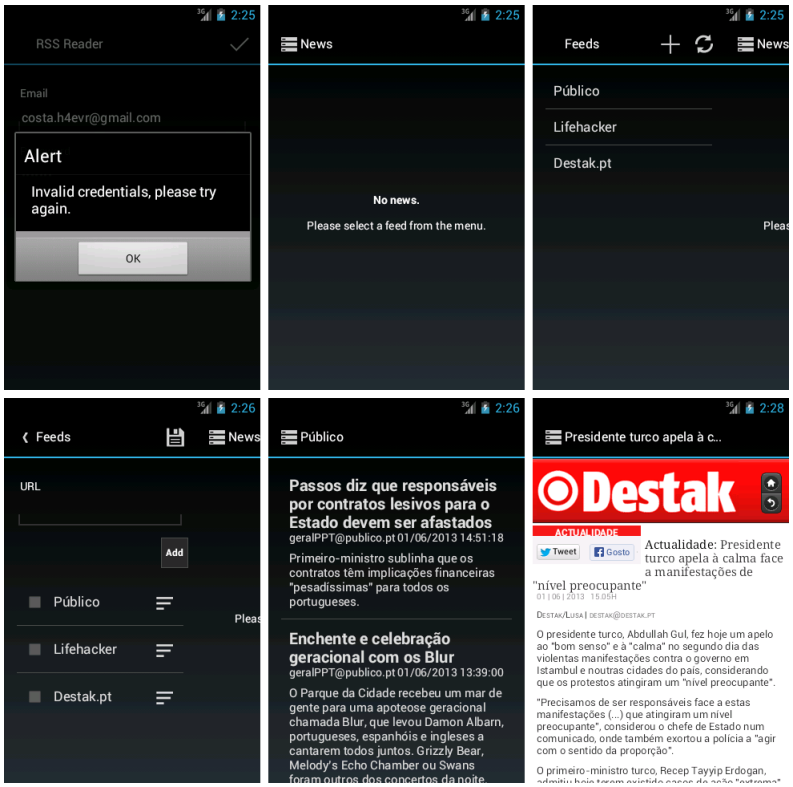


Figure 6.5: RSS Reader running on Android 4.2.2

After the application was developed, a deployment and packaging stage was implemented.
The deployment procedure includes minimizing and compressing all the Javascript code, so that the number of files to load and their size are reduced. To achieve this, the require.js optimizer, r.js, was used. The optimizer allowed to merge all modules into a single file and to run the uglifyjs program automatically, generating a file that could be imported using require.js.
However, due to the packaging system, that is described below, the load of dynamic Javascript files using require.js doesn't work so well, causing the files to not be loaded. So, to workaround

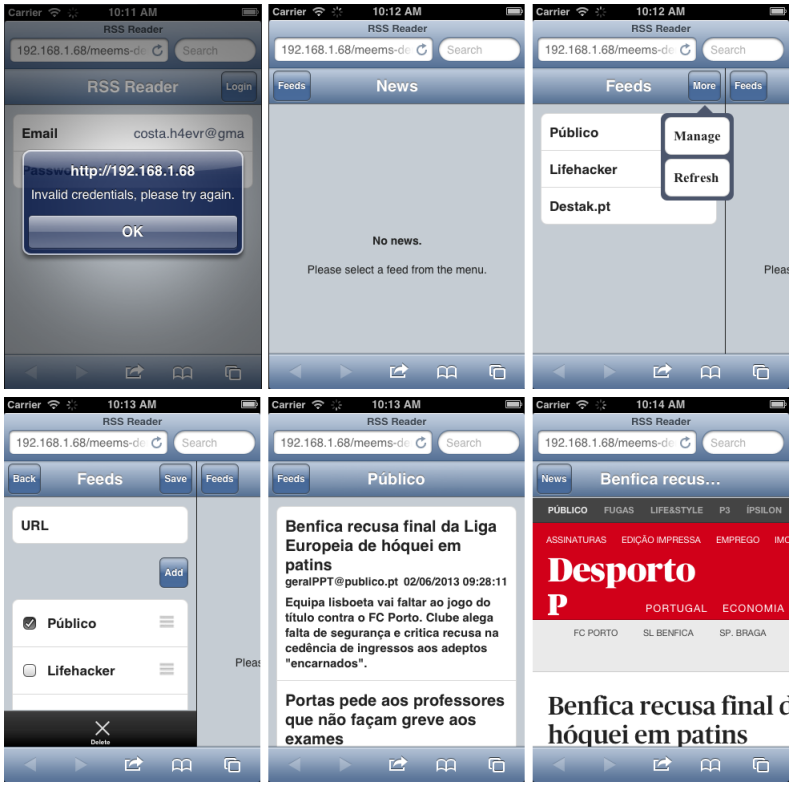


Figure 6.6: RSS Reader running on iOS Simulator, in Safari browser

this problem, a small "shim" loader was used: almond.js.[3] This loader replaces require.js and wraps itself around the minified modules single file, so that no dynamic loading of Javascript files is required, thus generating the final form of our application's logic.

The deployment routine above is just for optimizing the application's code to make it load faster. To distribute the application and really make it native-like, a native wrapper must be used. These wrappers generate a native application similar to a browser without any chrome and load the web application directly from the file system, which is normally the package itself. PhoneGap is one the most commonly used wrappers. However, using PhoneGap alone still requires to have a project per platform and deploy for each one of them. To overcome this and to build the final mobile application packages, the Adobe's PhoneGap Build[4] service was used. This service allows its users to provide their application's code and automatically compiles and generates native packages for six different mobile platforms: Android, iOS, Windows Phone, BlackBerry, webOS and Symbian. The application's code can be sent from the file system, or a Git or SVN code repository can be appointed. Since the RSS Reader application is available at GitHub[5], that very same repository was used.

Since building packages for the iOS platform requires a signing key, that only allows to test the package on a single device, the testing on the application on this platform was done using iOS

[3] https://github.com/jrburke/almond
[4] https://build.phonegap.com/
[5] https://github.com/h4evr/meems-demo-phonegap

simulator provided with Xcode. The Safari browser that the emulator brings has the same engine that PhoneGap uses on iOS platforms, therefore the behavior of the application is the same in both. For testing the Android package, the Android simulator was also used initially to speed up development. Later, the package was tested in the following devices:

- Nexus 7, running Android 4.2.1
  - **CPU:** Quad-code 1.2GHz Cortex-A9 (ARMv7)
  - **GPU:** ULP GeForce
  - **RAM:** 1GB
  - **Screen:** Capacitive 7" screen, 1280x800
- 2 Samsung Galaxy S Advance, one running Android 2.3.6 and other running Android 4.1.2
  - **CPU:** Dual-core 1GHz Cortex-A9 (ARMv7)
  - **GPU:** Mali 400
  - **RAM:** 768MB
  - **Screen:** Capacitive 4" screen, 800x480
- ZT-180 Upad, running Android 2.2
  - **CPU:** 1GHz Zenithink ZT-180 (ARMv6)
  - **RAM:** 256MB
  - **Screen:** Resistive 7" screen, 800x480

All devices were able to install and run the application, despite the different Android versions and specifications. The look of the application remained across all devices and so did the behaviour. The only performance issue that was noted was on the Upad, where the scrolling of long news list turned out to be slower than on the other devices. This indicates that some optimization is needed on the `List` widget to perform better on older devices.
See figures 6.5 and 6.6 for screenshots of the application running on Android and iOS.

6.4 Conclusions

Building an application combining the Meems Framework and require.js proved to be simple and fast, providing in the end a web application that looks almost native in Android and iOS platforms. Combined with Adobe's PhoneGap Build web service, creating hybrid applications for Android and iOS is very easy, just a commit to the repository, click on a "Update Code" button in their web page and wait for the packages to be built.
The performance on the devices tested was very satisfactory, but can be further improved to run better on older devices. Also, more effects should be parameterized to give a feel closer to native applications.

Developing a back-end server in node.js using the Express framework and MongoDB was a new experience. The way all the technologies involved speak the same language (Javascript and JSON) speeds up development and doesn't require the developer to be changing languages in the process. Node.js and Express make the separation of code and concepts easy, so the final source code is clean and the API is simple to understand.
This combination of technologies (Meems Framework, require.js, node.js, Express, MongoDB), all leveraging the power of Javascript and JSON, will allow developers to build full applications, from front-end to back-end, faster and better, using always the same language: Javascript.

Chapter 7

Conclusions

The market of smartphones is still evolving, as is the support for web technologies. The appearance of operating systems like FirefoxOS make web applications first class citizens, so in the future there will probably be more applications built using HTML5 and Javascript. And as the popularity of these operating systems grow, it is only natural that other platforms, like Android or iOS, also improve their support for these applications. The future of mobile development is currently too fragmented, with too many technologies, tools and frameworks, if one wants to target multiple platforms. Hopefully, in the future, when web mobile applications are more prominent, a cross-platform tool-set with well supported features will be developed, unifying developers and platforms.

Meanwhile, frameworks like Meems are a necessity. There are already many frameworks and libraries that help developers build their web mobile applications, but Meems fills a noticeable gap: the need to look native on each platform, with a single code-base. This goal was met, as demonstrated with RSS reader application that was developed to showcase the framework. Developing web applications still isn't easy as it should be, but using frameworks as Meems, the developer doesn't have to worry about how to render the user interface, instead he can focus on the content he needs to present to the user. As HTML5 support in mobile platforms grows, web applications will become more powerful and the frameworks that support them will need less boilerplate code, making them faster and allowing applications to access more features.

Besides the different support of the HTML5 standard on different platforms, one of the main challenges of developing cross-platform mobile applications nowadays is testing. The lack of appropriate tools and the discrepancies in the support of HTML5 by the different browsers makes it difficult for a developer to accurately test his application to ensure it works in all the desired platforms. One technique is to actually run the application in different smartphones and use remote debugging tools, like weinre or Chrome DevTools. As mobile operating systems evolve, it's expected a better native support for these tools. Adobe's PhoneGap Build allows one to enable debugging support in the developed applications, causing it to detect the applications that are running in the different devices. The user can then choose a session and a weinre debugger will be attached.

As new platforms appear, the mobile market evolves, along with how applications are developed. The future is still uncertain, but web applications will definitely play an important role in it.

7.1 Future Work

The Meems Framework provides a working basic framework, but it is far from complete. A project of this dimension would never be ended within the time frame of this dissertation, so its scope was reduced. But there is still much work to be done to turn it into a really useful and production-ready platform: more widgets, support for more platforms, better performance, integration with other tools, better user documentation with examples, tutorials and a community.

Since the framework is extensible, implementing new widgets and supporting new platforms is simple. Possible new widgets include: progress bar, picture gallery, comboboxes, dialogs and pop-ups. Existing widgets should also be improved and extended. For example, the `Aside` widget could be extended to be able to appear from either sides of the screen, instead of appearing only from the left. The `List` widget can also be further optimized, as it is certainly one of the most important widgets in mobile applications. Its performance varies according with the size of the list, which means that long lists degrade the performance of the whole application.

The support for other platforms, like Windows Phone, BlackBerryOS, FirefoxOS or Ubuntu Touch, will also be critical, as multiple platform support is one of the key features of frameworks of this kind.

As the frameworks evolves, the user documentation will also be made better and users are expected to collaborate in writing and translating it. A good solid documentation is an important step for any software that wants to be used properly and prosper.

Providing the final user with the best experience is the goal, but the Meems framework is also about making the life of developers easier. All the improvements above are certainly a step in this direction.

References

[ao13] amdjs organization. AMD. https://github.com/amdjs/amdjs-api, 2013. [Online; accessed 16-February-2013].

[App09] Apple. Apple Reports First Quarter Results, January 2009. http://www.apple.com/pr/library/2009/01/21Apple-Reports-First-Quarter-Results.html [Online; accessed 10-June-2013].

[App13] Apple. Start Developing iOS Apps Today: Set Up. https://developer.apple.com/library/ios/#referencelibrary/GettingStarted/RoadMapiOS/chapters/GetToolsandInstall.html, 2013. [Online; accessed 16-February-2013].

[Eva13] Benedict Evans. Facebook's 470m mobile app users. http://ben-evans.com/benedictevans/2013/1/2/facebooks-545m-mobile-app-users, January 2013. [Online; accessed 02-June-2013].

[Goo13a] Google. Android NDK | Android Developers. http://developer.android.com/tools/sdk/ndk/index.html, 2013. [Online; accessed 16-February-2013].

[Goo13b] Google. Android SDK | Android Developers. http://developer.android.com/tools/sdk/index.html, 2013. [Online; accessed 16-February-2013].

[Goo13c] Google. Policy and Best Practices - Android Developer Help. http://support.google.com/googleplay/android-developer/topic/2364761?hl=en, 2013. [Online; accessed 16-February-2013].

[Hen12] KeeKim Heng. Speeding up JavaScript: Working with the DOM. https://developers.google.com/speed/articles/javascript-dom, March 2012. [Online; accessed 04-May-2013].

[IDC13] IDC. Android and iOS Combine for 92.3Shipments in the First Quarter While Windows Phone Leapfrogs BlackBerry, According to IDC, May 2013. http://www.idc.com/getdoc.jsp?containerId=prUS24108913 [Online; accessed 10-June-2013].

[JA12] Jacky Nguyen Jamie Avins. The Making of Fastbook: An HTML5 Love Story. http://www.sencha.com/blog/the-making-of-fastbook-an-html5-love-story, December 2012. [Online; accessed 04-May-2013].

[jF+12] jQuery Foundation et al. jQuery Mobile Docs - Intro, 2012. http://jquerymobile.com/demos/1.2.0/docs/about/intro.html [Online; accessed 10-June-2013].

[Kan13a] David Kaneda. jQT (formerly jQTouch) — Zepto/jQuery plugin for mobile web development, 2013. http://jqtjs.com/ [Online; accessed 10-June-2013].

[Kan13b] Dan Kantor. The Story behind Exfm, April 2013. http://phonegap.com/blog/2013/04/23/story-behind-exfm/ [Online; accessed 05-June-2013].

[Mah13] Michael Mahemoff. Client Side Storage - HTML5 Rocks, 2013. http://www.html5rocks.com/en/tutorials/offline/storage/#web-storage [Online; acessed 10-June-2013].

[Nis13] Deep Nishar. 200 million members!, January 2013. http://blog.linkedin.com/2013/01/09/linkedin-200-million/ [Online; accessed 05-June 2013].

[O'D11] Jolie O'Dell. LinkedIn's new mobile app is so gorgeous, you'll actually want to use it, August 2011. http://venturebeat.com/2011/08/16/linkedin-mobile-app/ [Online; accessed 05-June-2013].

[O'D12] Jolie O'Dell. You'll never believe how LinkedIn built its new iPad app (exclusive), May 2012. http://venturebeat.com/2012/05/02/linkedin-ipad-app-engineering/#vb-gallery:1:421650 [Online; accessed 05-June-2013].

[O'D13] Jolie O'Dell. Why LinkedIn dumped HTML5 & went native for its mobile apps, April 2013. http://venturebeat.com/2013/04/17/linkedin-mobile-web-breakup/ [Online; accessed 05-June-2013].

[Ola12] Drew Olanoff. Mark Zuckerberg: Our Biggest Mistake Was Betting Too Much On HTML5. http://techcrunch.com/2012/09/11/mark-zuckerberg-our-biggest-mistake-with-mobile-was-betting-too-much-on-html5/, September 2012. [Online; accessed 02-Jun-2013].

[Sen13] Sencha. Sencha Touch - Build Mobile Web Apps with HTML5, 2013. http://www.sencha.com/products/touch/ [Online; accessed 10-June-2013].

[Thu13] Paul Thurrott. Why is the Facebook Beta App for Windows Phone 8 identical to the Android and iOs apps?, May 2013. http://winsupersite.com/windows-phone/why-facebook-beta-app-windows-phone-8-identical-android-and-ios-apps [Online; accessed 05-June-2013].

[Ton10] Tony Wasserman. Software Engineering Issues for Mobile Application Development. *FoSER*, 2010. http://works.bepress.com/tony_wasserman/4 [Online; accessed 23-February-2013].

[Wik12] Wikipedia. Approval of iOS apps — Wikipedia, The Free Encyclopedia. http://en.wikipedia.org/wiki/Approval_of_iOS_apps, 2012. [Online; accessed 16-February-2013].

Appendix A

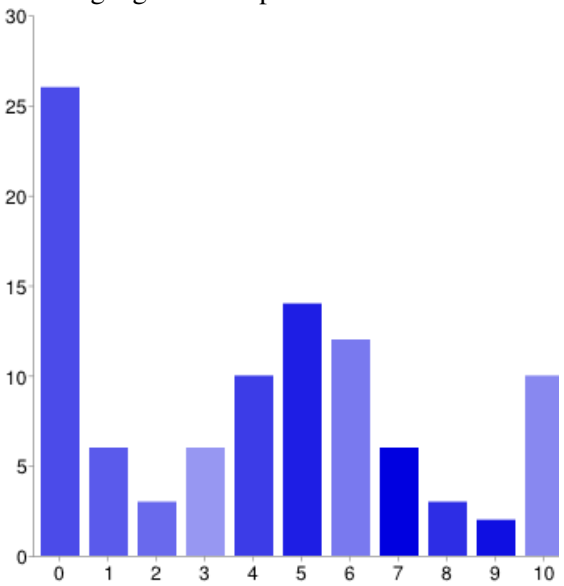
Survey Results

The results of the performed survey are shown below.

A.1 Experience

A.1.1 How many years of experience in software development do you have?

The language and the platform don't matter.

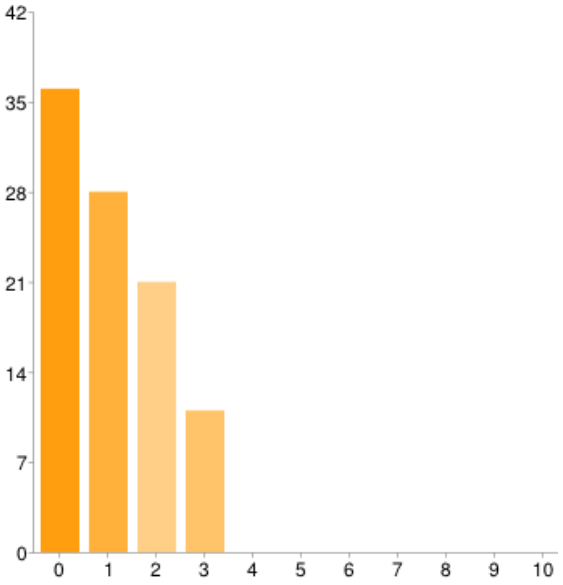


0	**26**	27%
1	**6**	6%
2	**3**	3%
3	**6**	6%
4	**10**	10%
5	**14**	14%
6	**12**	12%
7	**6**	6%
8	**3**	3%
9	**2**	2%
10	**10**	10%

This section's purpose it to collect the amount of experience you have developing software.

A.1.2 How many years of experience in software development do you have?

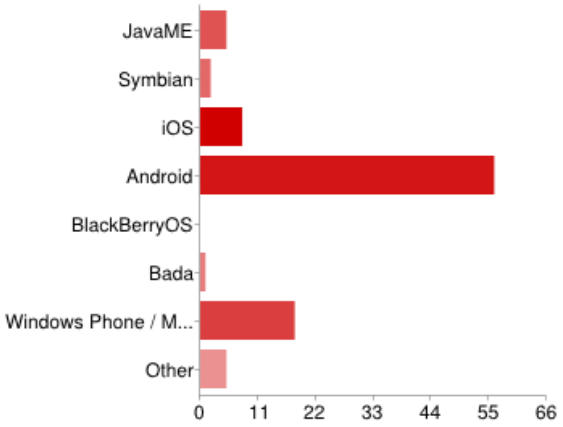
The platform doesn't matter.



0	**36**	38%
1	**28**	29%
2	**21**	22%
3	**11**	11%
4	**0**	0%
5	**0**	0%
6	**0**	0%
7	**0**	0%
8	**0**	0%
9	**0**	0%
10	**0**	0%

A.1.3 What platforms have you developed native applications for?

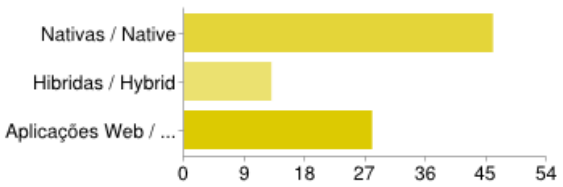
Choose all that apply.



JavaME	**5**	5%
Symbian	**2**	2%
iOS	**8**	8%
Android	**56**	59%
BlackBerryOS	**0**	0%
Bada	**1**	1%
Windows Phone / Mobile	**18**	19%
Other	**5**	5%

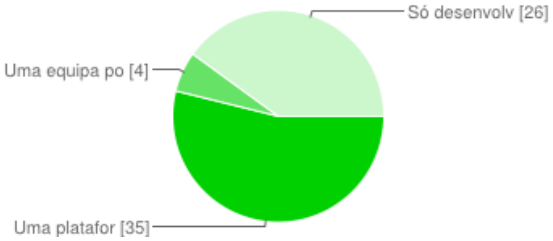
A.1.4 What kind of mobile apps do you develop?

Do you use the platform's provided SDK's (native), tools like PhoneGap or appMobi (hybrid) or do you develop web applications oriented towards mobile?



Native	**46**	53%
Hybrid	**13**	15%
Web App	**28**	32%

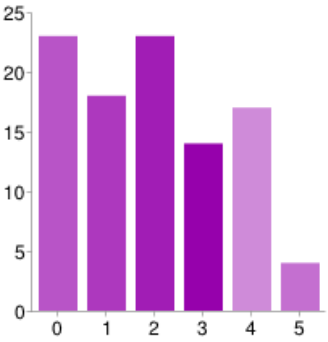
A.1.5 Do you target one platform at a time, or do you have a team per platform?



One platform at a time	**35**	54%
A team per platform	**4**	6%
I only target one platform	**26**	40%

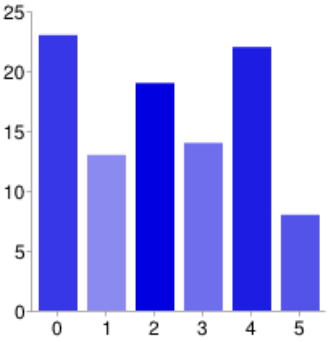
A.2 HTML5, Javascript and CSS3

A.2.1 What's your skill level in Javascript?



0	**23**	23%
1	**18**	18%
2	**23**	23%
3	**14**	14%
4	**17**	17%
5	**4**	4%

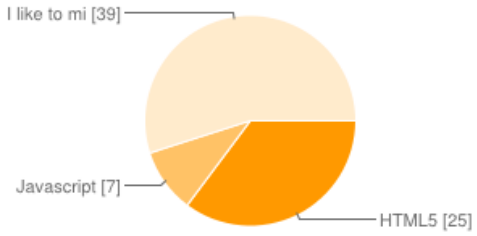
A.2.2 What's your skill level in HTML5 and CSS3?



0	**23**	23%
1	**13**	13%
2	**19**	19%
3	**14**	14%
4	**22**	22%
5	**8**	8%

A.2.3 Do you prefer HTML5 or Javascript for implementing user interfaces?

For implementing the views of your application, you prefer a library that enables you to use solely HTML5 (like KendoUI) to specify it or a library where you use Javascript to specify the GUI (like Sencha Touch)?

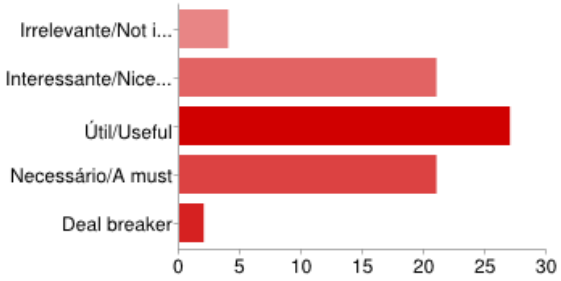


HTML5	**25**	35%
Javascript	**7**	10%
I like to mix HTML5 and Javascript	**39**	55%

A.3 Features

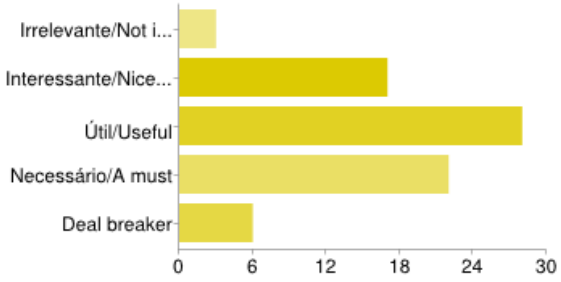
A.3.1 Classify the functionalities/characteristics below according to their importance to you

A.3.1.1 Analytics



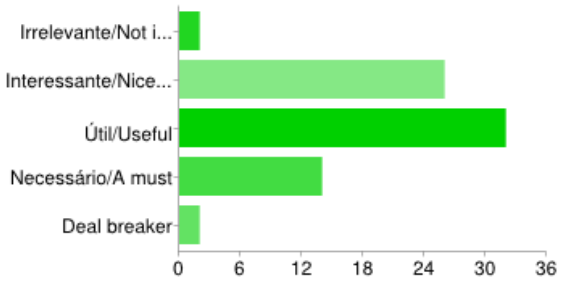
Not important	**4**	5%
Nice to have	**21**	28%
Useful	**27**	36%
A must	**21**	28%
Deal breaker	**2**	3%

A.3.1.2 Hardware acceleration



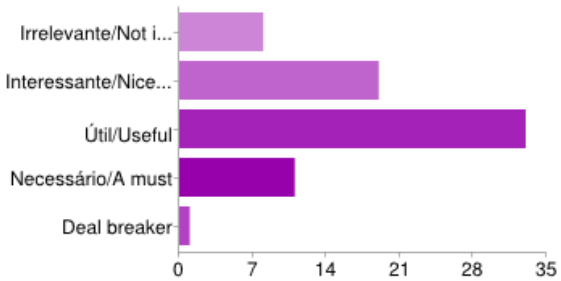
Not important	**3**	4%
Nice to have	**17**	22%
Useful	**28**	37%
A must	**22**	29%
Deal breaker	**6**	8%

A.3.1.3 Configurable effects



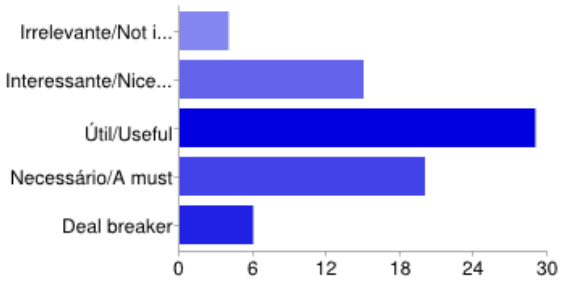
Not important	**2**	3%
Nice to have	**26**	34%
Useful	**32**	42%
A must	**14**	18%
Deal breaker	**2**	3%

A.3.1.4 Fixed scrolling emulation



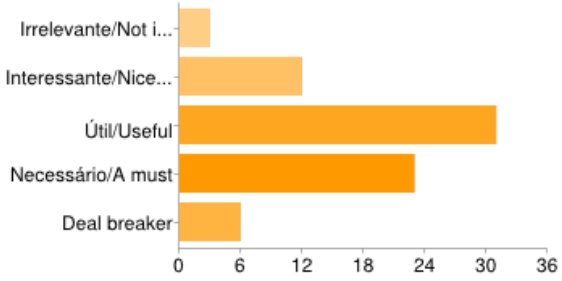
Not important	**8**	11%
Nice to have	**19**	26%
Useful	**33**	46%
A must	**11**	15%
Deal breaker	**1**	1%

A.3.1.5 UI Widgets



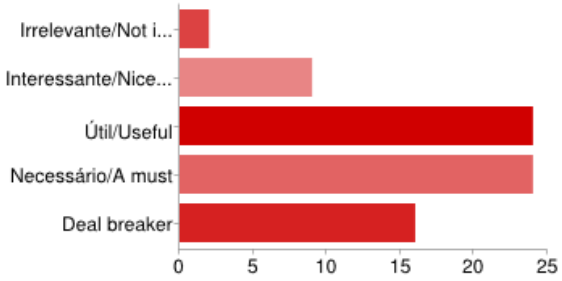
Not important	**4**	5%
Nice to have	**15**	20%
Useful	**29**	39%
A must	**20**	27%
Deal breaker	**6**	8%

A.3.1.6 Data bindings betweens views and models



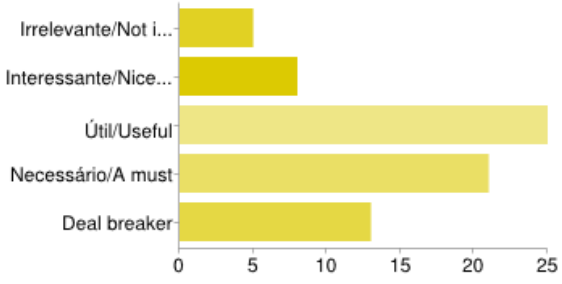
Not important	**3**	4%
Nice to have	**12**	16%
Useful	**31**	41%
A must	**23**	31%
Deal breaker	**6**	8%

A.3.1.7 Cross-platform support



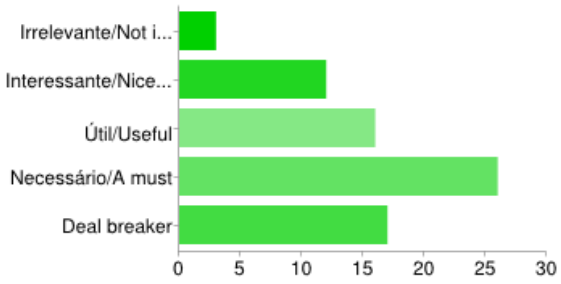
Not important	**2**	3%
Nice to have	**9**	12%
Useful	**24**	32%
A must	**24**	32%
Deal breaker	**16**	21%

A.3.1.8 Full SDK



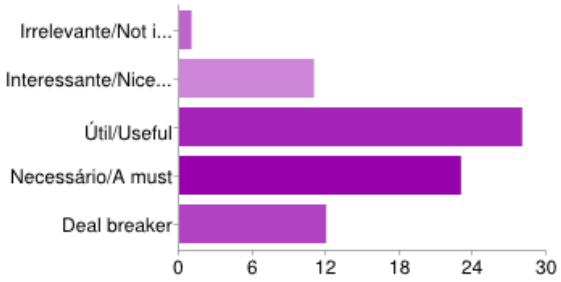
Not important	**5**	7%
Nice to have	**8**	11%
Useful	**25**	35%
A must	**21**	29%
Deal breaker	**13**	18%

A.3.1.9 Documentation



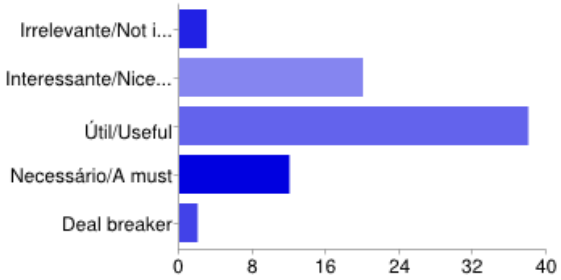
Not important	**3**	4%
Nice to have	**12**	16%
Useful	**16**	22%
A must	**26**	35%
Deal breaker	**17**	23%

A.3.1.10 Internationalization (i18n)



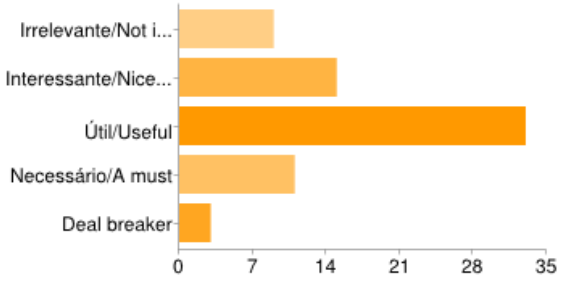
Not important	**1**	1%
Nice to have	**11**	15%
Useful	**28**	37%
A must	**23**	31%
Deal breaker	**12**	16%

A.3.1.11 Extensions



Not important	**3**	4%
Nice to have	**20**	27%
Useful	**38**	51%
A must	**12**	16%
Deal breaker	**2**	3%

A.3.1.12 Monetization



Not important	**9**	13%
Nice to have	**15**	21%
Useful	**33**	46%
A must	**11**	15%
Deal breaker	**3**	4%