Andrea Sánchez Blanco

DEVELOPMENT OF HYBRID MOBILE APPS

Using Ionic framework

Bachelor's Thesis
Information technology

May 2016

MAMK
University of Applied Sciences

DESCRIPTION

MAMK University of Applied Sciences	**Date of the bachelor's thesis** 23.05.2016
Author(s) Andrea Sánchez Blanco	**Degree programme and option** Information Technology

Name of the bachelor's thesis

DEVELOPMENT OF HYBRID MOBILE APPS
Using Ionic Framework

Abstract

The purpose of the current study is to analyse the advantages and disadvantages of web-based hybrid apps. This thesis explains from the beginnings of mobile phones how and why it has been created this solution, and how mobile's history has evolved to need an intermediate approach. It is also studied the general principles of good mobile user interfaces, in order to create a full good user experience from appearance to performance and usability, and how to implement the server side in an Ionic mobile application.

These have been done providing a general context about the first mobile phones to the latest ones, going through the different mobile OSs and focusing on an intermediate approach, hybrid web-based apps. After that I have explained Ionic Framework and the User Interface guidelines. Finally I have documented my practical part, associating it with the theoretical part, and explaining how I did the server side implementation on my app.

The conclusions I took are more general than I thought at the beginning. I have studied this solution with Ionic Framework, with which I have had an overall good experience. However, at the end I wondered if this intermediate solution is actually a solution or a patch for solving the developer problems, while hindering user experience.

Subject headings, (keywords)

Hybrid mobile apps, Ionic Framework, AngularJS, PhoneGap, User Interface, Client-server architecture

Pages	**Language**	**URN**
57	English	

Remarks, notes on appendices

Tutor	**Employer of the bachelor's thesis**
Matti Koivisto	Mikkeli Universtiy of Applied Sciences

CONTENTS

1 INTRODUCTION

The first hand-held cell phone was introduced in 1973 by John Mitchell and Dr. Martin Cooper, weighting 0.8kg. Ten years later, in 1983, the first commercially available mobile phone was released, nicknamed “brick phone” because of its big dimensions. This was a hard beginning, focused on business functions and with the basic idea of being able to call without cables. Soon, in the 90’s, Nokia started to develop smaller phones and with customizable cases. Since then, mobile device market has not stopped growing and becoming more and more important in our daily lives.

The last big improvement in the mobile world was the release of iPhone in 2007, initiating the touchscreen devices with an advanced OS and the app stores. These devices were called “smartphones”. Because they were able to make more than the basic phone functions like calling and sending SMS, it was possible to play music, take photos and surf on Internet without any buttons. This was revolutionary, consequently in only few years most people got a smartphone. However, iPhone was quickly catched up by other brands, dividing the current smartphone market into two main operating systems: iOS and Android. Other mobile operating systems like Windows phone are also gaining importance.

This OS division causes that every app must be developed for each platform in each programming language, which requires a lot of time that most of the companies do not have. Since smartphones have become so important, enterprises want to have their apps available for as many people as possible, which means the development of, at least, one iOS and one Android app with the same content. As a solution, hybrid apps were created, taking advantage of the best of HTML5 to be able to create cross-platform apps. However, the cross-platform solution has its disadvantages, as using native device controls have higher performance than web apps.

The cross-platform solution is the one explored in this thesis, by explaining how hybrid apps work and the different available ways to develop them. The theoretical aim is to compare hybrid apps with native ones (Chapter 2), what different possibilities exist in the hybrid development and the basics of frameworks (Chapter 3). As practical work, the Ionic framework is the one selected to carry out this hybrid app study. Therefore, it is also explained in depth why I selected this one and how it works (Chapter 4). Design patterns in mobile development are

also part of the theoretical aim, because they help to create friendly user experience apps, which nowadays is a very important part of app development to take into account (Chapter 5).

Related to the practical aim it is explained how to install Ionic on Windows for the development process, the structure of my app with some diagrams, the UI design patterns applied, the performance issues I faced and how to solve them, and the server connection (Chapter 6). The app consists of a home screen where the user chooses whether to search recipes based on ingredients or categories or whether to have a random recipe pick. This is made to implement both, the user interface guide lines and the Ionic development, as a possible solution in the cross-platform web mobile development. Finally, last chapter discuses the conclusions (Chapter 7).

2 MOBILE DEVELOPMENT

The purpose of this section is to introduce when the first mobile phones appeared, how they have evolve and where are we now. It is also important to see the variety of mobile operating systems that coexist nowadays, in order to understand why there is a necessity of bringing them together making easier the mobile app development process. This hybrid solution is also introduced on this section, comparing both upsides and downsides.

2.1 Beginning of cell phones

The Motorola DynaTAC 8000X was the first commercially available cell phone, marketed in 1983 and offered commercially in 1989. It weighted 800 grams and the autonomy was 30 minutes talking and around 10 hours of charging. It was called “the brick” because it had the size of a brick, it made calls and there was a simple contacts application included in the operating system (Ismash 2015).

The first generation of mobile phones was designed and developed by the handset manufacturers. The competition had many secrets and they did not want to share their internal work and that is why the development of software was made inside each company. In the 90’s Nokia released the first mobile with a famous “time-waster” game of the 70’s, Snake. Since

then, mobiles began to include more games and people changed their idea about them, mobile phones could not only make calls, but also entertain people. (Clarck, 2015.)

Related to this improvement came up a problem, manufacturers wanted to provide more services to each user, and this leads to Wireless Application Protocol (WAP). WAP was a stripped-down version of HTTP, which is the backbone protocol of the Internet. Unlike traditional Web browsers, WAP browsers were designed to run within the memory and bandwidth constraints of the phone. This seemed to be the solution, but WAP browsers were slow and frustrating. Typing long URLs was horrible and critics started to call WAP "wait and pay". (Clarck, 2015.)

Handset manufacturers realized that they should start to be less protectionist with their policies and expose their internal work to some extent. Thanks to this, different mobile platforms emerged and developers began and are still developing apps for them.

2.2 Mobile platforms

Nowadays there are eight mobile operating systems in use, each one with their own characteristics and implementations. In order to be able to compare them, they must be first described shortly one by one before a conclusion can be explained.

Android

Android is based on the Linux kernel and currently developed by Google. It is primarily for touchscreen mobile devices such as smartphones and tablets, and it is under open source license. The last version, Android 6.0 Marshmallow, was released on October 2015. However, the next-to-last version Lollipop, is the largest and most ambitious release on Android.

Android Lollipop and Marshmallow have a redesigned user interface built around a responsive design language called Material Design. Other changes include improvements in the notifications and support for 64-bit-architectures. Furthermore, Google replaces the virtual machine Dalvik with the Android Runtime (ART) to improve the application performance and optimize battery usage, known as Project Volta (Chester 2014).

iOS

iOS, originally iPhone OS, was created and developed by Apple Inc. and distributed exclusively for Apple hardware, like iPhone, iPad and iPod touch (Wikipedia 2015a). It is based on the XNU kernel (*X is Not Unix*) written in C/C++.

iOS works as an intermediary between hardware and applications. iOS architecture is built on a set of layers, namely Cocoa touch, Media, Core services and Core OS. Cocoa touch layer is responsible for the appearance of the application, Media layer provides the graphics, audio and video services. Core services layer provides the key services for the application like social media, networking and iCloud, and Core OS is the low level layer that deals with low level services. The latest version is iOS 9. (Manishankar 2014.)

Windows

Windows Phone was developed by Microsoft and written in C/C++, replacing Windows Mobile and Zune. It features a user interface derived from the Metro Design language. Windows Phone 8.1 uses Windows NT (sharing components with Windows 8) replacing the Windows CE kernel.

Windows 10 Mobile is the latest iteration of the Windows operating system, now unified with the Windows Phone OS. It was released on February 12, 2015. It is the first attempt by Microsoft to unify their desktop, tablet and phone operating systems into a single OS. Windows favors writing applications in C# but it is also possible to use C++. Applications run on a light .NET version. (Windows Central 2016.)

Blackberry OS

Blackberry OS was developed by its manufacturer RIM. Blackberry OS is written in C++ and based on QNX Neutrino RTOS kernel. The OS is proprietary and only used in the line of Blackberry smartphones. It is known for its security, multitasking capabilities and interoperability with corporate email infrastructures. (Crackberry 2015.)

Firefox OS

Firefox OS is a Linux kernel-based and open source system developed by Mozilla. It uses open standards and it puts emphasis on HTML5 technology to go along with device capabili-

ties. It is also written in CSS, JavaScript and C++. The first phone with Firefox OS was ZTE One in 2013 and the latest version is Firefox OS 2.1. (Developer mozilla 2015.)

Sailfish OS

Sailfish OS was founded by Nokia employees and now it is being developed by Jolla in cooperation with the Mer project community and corporate members of the Sailfish Alliance. It is open source and combines the Linux kernel, Mer core provided by MeeGo and proprietary software by Jolla. The distinguishing features are Android compatibility, gesture-based interface and shortcuts, which means that Jolla devices are able to install Android applications and the user experience is based in gestures like swiping, pressing, tapping and dragging. (Eadicicco 2013.)

Tizen

Tizen is based on the Linux kernel and the GNU C Library. It is written in HTML5, C and C++. It is a project within the Linux Foundation and managed by Samsung and Intel among others. It targets a very wide range of devices including smartphones, tablets, wearable computerized products and smart home appliances. (Wikipedia 2015b.)

Ubuntu Touch

Ubuntu Touch is the mobile version of Ubuntu OS developed by Canonical, designed for touchscreen devices. It is based on the Linux kernel and written in C++ and QT5 for the user interface. It is gesture-based and the applications are still in development, because there is not a lot of variety and some of them are web apps, which are far from native performance. (Ubuntu site 2015.)

There are also several discontinued mobile OS: Symbian, Windows mobile, Bada, Palm OS and webOS. They are the result of the first attempts to create smarter phones, not failed projects but intermediate steps to achieve better mobile OSs.

To conclude, each platform is based on different kernels, programming languages and user interfaces, which means that the app development process for each one differs in too many aspects, involving different IDE's, simulators and user interface frameworks. This leads enterprises to choose one or few platforms to develop their app, because creating an app for every OS would cost an amount of money and time that not all business can afford. This issue

makes us think that one app working in all devices would be a really good solution, learning only one programming language for all, “write once, run anywhere”, as Java slogan says.

2.3 Mobile applications

The word application was traditionally a generic term for any standalone bit of software that runs on top of a computer’s operating system, like iTunes runs on top of Mac OS (Gahran 2011). A mobile application, or mobile app, is software designed to run on handheld devices, such as smartphones or tablets, and that can connect to wireless networks. They can be pre-loaded on the device, downloaded from several app stores or from the internet, or made by users themselves.

2.3.1 Types of mobile applications

Usually when people say “mobile app”, they refer to a native app. The following explains the differences between the three types of mobile applications: native apps, web apps and hybrid apps.

Native applications

Native apps are developed for a particular platform or device. They are coded in a specific programming language, such as Objective C for iOS or Java for Android. Because of this, they can interact with and take advantage of the operating system features and other software that is typically installed on the platform and they also have the ability to use device-specific hardware and software, meaning that native apps can take advantage of the latest technology available. In addition, users can also run the apps without Internet connection. (Rouse 2013.)

A native app is installed directly on a mobile device and developers must create a separate app for each mobile device. The data can be stored on the device itself or remotely, and be accessed by the app. Depending on its nature, Internet connectivity may or not be required. (Rouse 2013.)

Web applications

Web apps are stored on remote servers and delivered over the internet. They are not real apps, in fact, they are websites that *look and feel* like native apps. They run on browsers and are

usually written in HTML5. Users first access them as they would access any web page: They navigate to a special URL and then have the option of installing them on their home screen by creating a bookmark to that page. Nowadays, as more and more sites use HTML5, the distinction between web apps and regular web pages has become blurry. (Ali 2013.)

Hybrid applications

Hybrid apps are a mixture of native and web apps, because they are written in HTML5 like web apps, but they are also installed and they can access the hardware of the device like native apps. In other words, they are web apps hosted inside a native application that utilizes a mobile platform's WebView to run and process the JavaScript. For clarification, a WebView is just a browser bundled inside of a mobile application.

Hybrid mobile applications are built in a similar manner as websites. Both use a combination of technologies like HTML, CSS and JavaScript. However, instead of targeting a mobile browser, hybrid applications target a WebView hosted inside a native container. This enables them to access hardware capabilities of the mobile device. (Bristowe 2015.)

2.3.2 Comparison between native and hybrid apps

Between the three types of mobile apps I will only compare native and hybrid, because web applications are clearly categorized to have very low performance that cannot access many device specific functions. Therefore, they do not represent a real opponent to native apps, as hybrid does. The first step to compare native and hybrid apps will be to point out the advantages and disadvantages of each one, to obtain the conclusions.

Starting with native apps, they have unrestricted access to the platform with specific UI. They use the last available hardware resources to improve performance, having smoother experience. They can work in offline mode and send push notifications and reminders to users. (Sundqvist 2013.) However, they have to be written for each platform and developers need to know the programming languages and be familiarized with each IDE, as shown in Table 1.

TABLE 1. Language and IDE per platform

Platform	IDE	Language
iOS	Xcode	Objective-C and Swift
Android	Eclipse, Android Studio, etc.	Java
Windows Phone	Visual Studio	C#

Taking iOS as an example, the developer, first of all, needs to own a Mac and have the Developer Account (EUR 87 / year), also know Objective-C, Xcode and the iOS development guides. This is just one platform, therefore developing for all would require a big budget to support all platforms. And, if becoming familiar with the OS is needed, the learning curve in native apps is the highest of the three type of apps, involving long development cycles.

On the other hand, hybrid apps have a medium-fast development process, requiring to know only the web languages (HTML5, JavaScript and CSS) and the specific mobile development framework (like PhoneGap). One only needs to create one app for all the supported platforms. They can access native layers and also work in offline mode. The disadvantages of hybrid apps are that they take a lot of experience and knowledge to build one that is near native performance. Cross-browser issues still exist, and they do not always look or feel native. Native apps talk directly to the operating system, while hybrid apps talk to the browser, which talks to the OS. Thus, there is an extra layer hybrid apps have to pass, and that makes them slightly slower and coarser than native apps. This problem is unsolvable. (Quirksmode 2015.)

In conclusion, native apps are better when the goal is performance and the time and budget does not matter. They are compiled into machine code (for example, Dalvik byte code under Android), which gives the best performance you can get from the mobile phone (Ziflaj 2014). And, hybrid apps are preferable when fast development, easy maintenance and small budget are the limitations.

3 CROSS-PLATFORM MOBILE FRAMEWORKS

Now that all the mobile OS diversity has been explained with its pros and cons in Section 2, this Section is a more concrete explanation of the hybrid approach. I will start defining the word framework, as it is a key word during the rest of this document, and I consider important

to first give the description in order to have in mind the right idea. After that, I will introduce the two hybrid solutions that are actually developed. Finally, I will talk about all the possible solutions I have considered before taking a decision on the hybrid development, before going ahead with the Ionic Section.

3.1 What is a framework?

In general terms, a framework provides useful tools that facilitate the building process of programs for specific platforms. Software frameworks may include support programs, code libraries, compilers, tool sets, and application programming interfaces (APIs) that bring together all the different components to enable development of a project or solution. (Niagaraax 2015.)

In a mobile oriented view, cross platform web-based mobile frameworks are designed to support the development of phone applications that are written in HTML5 and leverage native phone capabilities.

3.2 Cross-platform alternatives

For cross-platform development there are many possible options to choose, nevertheless the basic idea is clear, minimize the development effort reaching as many platforms as possible, without straying too far from native app performance. The choices can be divided into two main categories: web-based frameworks and mobile cross-platforms tools.

3.2.1 Web-based frameworks

Web-based frameworks wrap web apps as native apps. They are built using HTML, JavaScript and CSS in order to run in many platforms using the web browser, and they also take advantage of the device features, getting closer to native apps. In fact, web-based apps are the real concept of hybrid apps because they are an intermediate approach between the web and the native apps.

One of the most popular and basic frameworks is PhoneGap because it can be combined with many UI frameworks and it is almost the only one who provides a native wrapper to compile

the web applications into native apps. Most of the frameworks need it, so it is important to talk about its functionalities.

PhoneGap / Cordova

PhoneGap is a set of device APIs created by Nitobi in 2009, purchased by Adobe Systems in 2011 and later donated to Apache Software Foundation, starting the open-source software Apache Callback, now renamed as Apache Cordova. PhoneGap is a distribution of Apache Cordova, but currently, the only difference is in the name of the download package and will remain so for some time (LeRoux 2012). PhoneGap enables the building of mobile device applications using JavaScript, HTML5 and CSS, creating hybrid apps. Currently it supports iOS, Android, Blackberry, Windows Phone 8, Ubuntu, Firefox OS and Tizen.

PhoneGap is not just a native wrapper of a web app, it provides a bridge for Javascript code to call native device features (like Camera, Digital compass, Microphone, etc). These are provided via PhoneGap plugins that are re-implemented with identical Javascript interfaces on each OS platform (O'Hagan 2014). There are also many other third part plugins that allow extra functionalities, including native sharing and analytics. However cross-platform support is typically more limited with third-party plugins.

PhoneGap combined with a UI framework is a powerful tool to develop hybrid applications, and that is the important step in which spend time and choose one. Sometimes the UI framework is loaded into the PhoneGap project, like jQuery mobile, and other times it is a complete framework that integrates Cordova, like Ionic Framework.

3.2.2 Cross-platform tools

Cross-platform mobile tools create native apps, not hybrid apps, but with the difference that they are focused on having an application working almost the same way on different platforms. They take advantage of a core of mobile development APIs which can be normalized across platforms. These areas should be targeted for code reuse, but they also have platform-specific APIs, UI conventions, and features which developers should incorporate when developing for that platform. Platform-specific code should exist for these use cases to provide the best possible experience. (Whinnery 2012.)

An important mobile cross-platform tool is Appcelerator Titanium, which is an attempt to achieve code reuse with a unified JavaScript API, with platform-specific features and native performance to meet user expectations. Titanium application are written as native applications in JavaScript. Titanium is a framework for writing native apps, versus an abstraction from the actual targeted platform. (Whinnery 2012.)

3.3 UI / Development frameworks

The process to select one framework is arduous and it takes time because of the research, comparison and choosing process. As hybrid apps are a new concept, there are many websites comparing UI frameworks but not so many helping in the development of the app itself using the framework, so that is an important point to consider. Working with new tools not only means working with the latest technologies, but also working with beta or unstable versions, having to deal with unsolved problems, or with limited information even from the official webpages.

At the same time as looking for the current UI frameworks, I made a list with the most attractive ones and that look good to work with. In my list I had the following UI frameworks: jQuery mobile, Onsen UI, Ionic, Mobile Angular UI, Chocolate chip UI, Sencha Touch and Framework 7.

jQuery mobile is the first to appear in the list because I have already worked with it and it is the most popular UI mobile framework, therefore, it has a lot of information and support on Internet. It is a HTML based user interface system with many component, features and support. It is designed to make responsive web sites and apps that are accessible on all mobile devices. (jQuerymobile 2016.)

Onsen UI works with Angular JS and supports jQuery to build a PhoneGap / Cordova app, focused on performance and ease of use. It supports Android, iOS, Firefox OS, Windows Phone 8.1 and desktop browsers. It has an HTML5 IDE called Monaca IDE, good official documentation, it is built around Topcoat CSS library and support native looking themes for Android and iOS. (Gaić, 2014.)

Mobile Angular UI uses Bootstrap and Angular JS, so like they say in the website, “anything that supports Bootstrap 3, Angular 1.3 and CSS3 transitions and transforms should fit to Mobile Angular UI”. (MobileAngularUI 2015)

Chocolate chip UI uses standard web technologies and it is built on top of jQuery library. It supports iOS, Android and Windows Phone with a custom CSS theme for each one, giving a native look & feel. It is a light and customizable framework that can be use alone or with Phonegap. On the other hand, it is more focused on iOS than on Android or Windows Phone, and it has small community.

Sencha Touch uses Ext JS, a JavaScript library developed by Sencha to build interactive web apps. It is based on a Model-View-Controller design patter and also focused on native-looking and high performance. It supports iOS, Android, Windows Phone, Tizen and Blackberry platforms. It provides majority of the components and widgets.

Framework 7 is a free and open source framework focused on iOS and Android. It provides a full set of UI components that are exactly the same expected when opening an iOS or Android (with Google Material design) applications.

Ionic Framework is an open source framework that wraps AngularJS to provide the application structure and Ionic itself focuses on the user interface. As it uses Angular, Ionic is a MVC framework, separating presentation from logic while boosting maintainability and productivity, and uses jqLite. It is focused on performance, it has a powerful CLI to create, build, test and deploy the apps, and good-looking components. The website documentation is clearly organize, with a specific forum for Ionic developers and more community in other pages like StackOverflow, so dealing with problems should be better to face. Ionic supports Android and iOS, and Ionic 2 will also have support for Windows 10 Mobile.

In Table 2 I have compared different UI frameworks by listing their main advantages and disadvantages. Because of its benefits and lack of disadvantages I chose the Ionic framework and decided to test it by myself in the hybrid development. In the next chapter I will introduce the Ionic framework in more detail.

TABLE 2. UI Frameworks comparison

UI Framework	Advantages	Disadvantages
jQuery	Many components.& features Good support Good documentation	Not the best performance
Onsen UI	Uses AngularJS Focused on performance Supports many OS Good documentation	Small community only in StackO-verflow
Mobile Angular UI	AngularJS used	Few information Few support Unclear platforms compatibility
Chocolate Chip UI	Native look & feel for each platform	Focused on iOS Small community Few components
Sencha Touch	Based on MVC Focused on native look and performance Supports many platforms Many components	Not open source, payment frame-work
Framework 7	Native look & feel for both platforms Material design on Android Many components	Focused only on iOS and Android
Ionic	AngularJS used Good looking components Clear documentation Large community and Ionic forum site	

4 IONIC FRAMEWORK

Ionic is a hybrid app development framework product of Drifty, a Silicon Valley startup. It is a young framework whose first alpha release was in late November 2013, and after 14 betas and 5 RCs (release candidates), the final release was launched on 12th May 2015 as Ionic 1.0.0 uranium-unicorn. Ionic is based on ngCordova, their own project, which the ngCordova web-site (2015) defines basically as "Cordova with the power of AngularJS", combining both and making possible to build Cordova hybrid apps with the help of AngularJS. Ionic also supports Syntactically Awesome Style Sheets (SASS) CSS extension. It is focused on being optimized for touch devices (rather than mobile websites), and because jQuery Mobile is now considered as a low performer, Ionic uses jqLite in case of need to access the DOM, without loading all jQuery, thus using less memory.

As it is built using AngularJS I will first explain what this framework is, its main features and best practices, because creating good quality AngularJS code is crucial for creating an Ionic App. Then I will explain what ngCordova is, and why it is important on Ionic. Finally, I will go through the particular features that Ionic brings on top of all of this.

4.1 AngularJS

AngularJS is a MVC JavaScript framework for Web Front End development for adding interactivity to HTML. It helps in the creation of Single-Page Applications (SPA). SPAs are websites that fit on one web page with the aim of having a more fluid user experience in desktop apps. It was created in 2009 by Google, but became popular at the end of 2012.

Before Angular appeared, front-end applications used jQuery (among others, like Mootools) to help us in the client JavaScript code. It was easier to manipulate the DOM, add effects, AJAX and more, but there was no pattern to follow. All the functions where created when needed and added to the code, creating the so called Spaghetti Code, and becoming hard to handle. When frameworks started to implement the MVC pattern, concepts became easier to separate and manipulate.

AngularJS extends HTML attributes with Directives, and binds data to HTML with Expressions (w3schools 2015). It includes jqLite (jQuery Lite), which is a subset of jQuery, a

library with DOM access and the minimum functionalities. This means that some Angular directives use jQuery, but if the programmer needs the full jQuery, it is also possible to load it, and AngularJS will stop using jqLite. AngularJS can be loaded into a project adding the following line of code:

```
<script
src="http://ajax.googleapis.com/ajax/libs/angularjs/1.3.14/angular.m
in.js"> </script>
```

In the next pages I will explain the main concepts of Angular and the components it uses to have later a good understanding of the Ionic development process. It is the very basics of Ionic. It has many important concepts to go throw, and it is directly related with the Ionic development. For this reason, I consider clearer to introduce first with AngularJS concepts before Ionic features.

4.1.1 Model-View-Controller

Model-View-Controller (MVC) is a software architecture pattern used to implement some user interface systems. It was developed, because there was a need for stronger software with a better life cycle where the maintainability is easy and the code reuse and concept separation are strengthened. Code school (2015) separates the domain/application/business logic from the rest of the user interface into three parts:

a) Model: It is the stored data in a database or XML with the business rules that transform that information (taking into account the user actions).
b) View: It is the HTML page.
c) Controller: It is the code that obtains data dynamically and generates the HTML content.

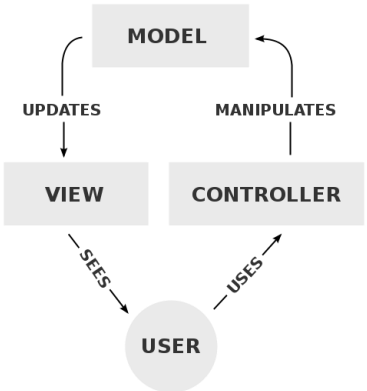

FIGURE 1. MCV component interactions (Code school, 2015)

Figure 1 shows the collaboration of the MVC components. When the user makes an action in the view, the controller acts and updates the model, the model recognizes this and updates the view, showing the new data to the user.

4.1.2 Features

In order to use Angular the developer must know HTML, CSS and JavaScript. Angular helps to organize the JavaScript code, to create **responsive** (as in fast) **websites**, and it is easy to test, creating maintainable software. I consider important to define properly what "responsive" means, because it is a significant feature and improvement that Angular provides.

When a browser initiates the request to a server, the server answers with the webpage and assets (HTML & JavaScript), and the browser loads up the entire webpage. When the user clicks on a link, in a traditional page-refresh response cycle, the browser initiates a new request, which the server will answer with a new webpage and assets, loading again the whole site (Figure 2).

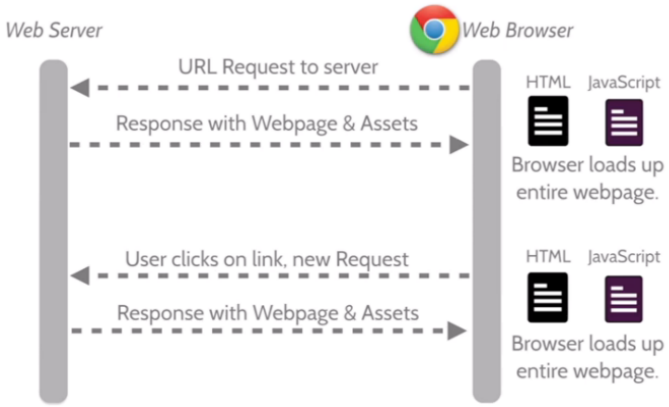

FIGURE 2. Traditional page-refresh (Code school 2015)

In the responsive page-refresh, the server is only going to give the information the user needs to update the page, JSON data. When the browser has this data, it is going to load it on the existing page, updating what we see on our browser (Figure 3). (Code school 2015.)

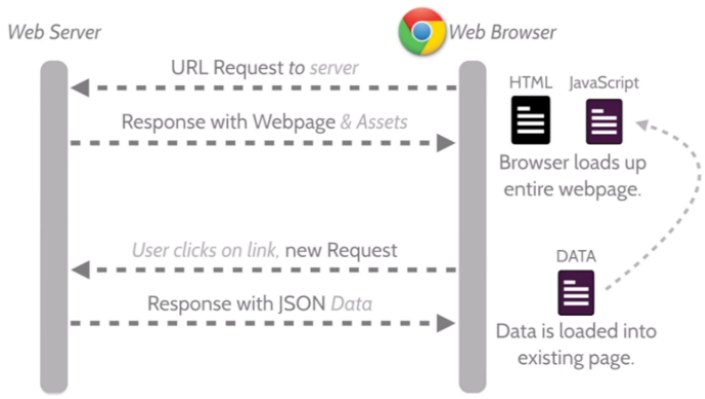

FIGURE 3. Responsive page-refresh (Code school 2015)

The responsive refresh-page is a more efficient and faster way than the traditional, because it does not refresh the whole page each time that a new request is made. In addition, most of the templating systems bind data only in one direction, merging the model and the template into a view. After this, changes in the model are not automatically reflected in the view. This is called one-way data binding, as shown in Figure 4.

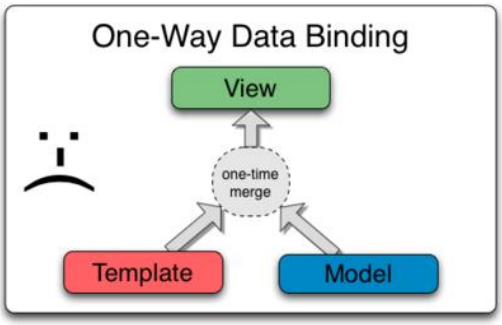

FIGURE 4. One-way data binding (Code school 2015)

Two-way data binding is constantly updating the changes in the model to the view, and vice versa (Figure 5). It simplifies the programming model for developers, because they do not have to manually write code that the view constantly syncs with the model, and the other way around, as it happens in the classical data binding. Angular data binding markup is done with expressions inside double braces {{ }}.

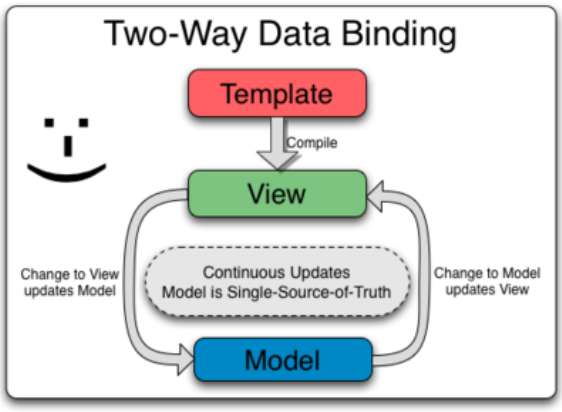

FIGURE 5. Two-way data binding (Code school 2015)

Angular is **REST-friendly**, which is a software architecture consisting of guidelines and best practices for creating scalable web services. AngularJS has **deep linking,** setting up a link for any dynamic page. Any page that is not the home page is considered ”deep”, and therefore deep linking consists of using a hyperlink that leads to a specific piece of web content on a website rather than the home page. **Form validation** is also an important characteristic in Angular, because it has many directives available to make form validations easier than in traditional web development. It also provides **Localization**, adapting applications and text to enable their usability in a particular cultural or linguistic market. (AngularJS official documentation 2015.)

The last but not least important is **dependency injection**. It is the way of specifying what the controller needs to inject the dependencies, in Figure 6 a Service, as arguments.

FIGURE 6. Dependency Injection

To conclude, all these features are very useful, because they allow you to create more complex apps, thanks to the REST-friendly and deep-linking features, having several pages in one app with clear navigation. Form validation and localization make the programmer’s work easier,

by automating processes that are not necessary complicated, but take time to write them. Finally the dependency injection is a big improvement, because it allows isolating concrete implementation that, if they are no longer needed, are easy to remove from the project, which means also easier testing process and reduces boilerplate code (sections of code that have to be included in many places).

4.1.3 Main concepts

As described above, AngularJS gives a structure that encourages a separation of concerns, organizing the code into controllers, services, directives and filters, while maintaining implementation flexibility and extensibility (Lamb 2015). In the next paragraphs I will explain their basics to understand when to use them and how they improve the app development. This text is mainly based on Natividad, frontendlabs.io webpage (2015).

Directive

A directive is a marker on a HTML tag that tells Angular to run or reference some JavaScript code. AngularJS provides many default directives with functionalities, but the programmer can also write their own directives to satisfy their needs.

Filter

Filters are responsible for transforming data to show it as requested. AngularJS has some native filters, like lowercase, uppercase, limitTo, among others. They are used with the "|" symbol, as shown in the example below:

```
<li ng-repeat="item in items">{{item.name | uppercase}}</li>
```

Expressions

Expressions are used to insert dynamic values into the HTML, for example:

```
<p> I am{{4 + 6}} </p> → It will show "I am 10"
```

Scope

Scope is a really important concept in AngularJS, because it connects the data between the controller and the view (two-way data binding). It is an object which can be set with data and also methods.

Modules

Modules are the containers of the different parts of the app with an identifier name and the dependencies. Modules are essential in Angular development, because they organize the code and introduce good practices, like being isolated containers to avoid collisions between JavaScript scripts (Hurtado & Jorge, 2014). The most basic module is, when creating an app with Ionic. The entire app itself is a module.

In order to make them work they must be linked with the view using ng-app directive, the syntax used to declare them is using the Angular method .module():

```
<body ng-app="myApp">
    ...
</body>
angular.module('myApp(module name)', []);
```

This Angular method returns a module object, which also has several methods like config, run, provider, service, controller, etc. that are used to control the presentation and business logic of the app. Below are explained Controller and Service methods because they are the basic ones and more often used. The idea is not go throw all the methods that AngularJS has, but know the more important ones.

Controller

Controllers are objects that contain the presentation logic of the app defining functions and values, they implement the functionality of the view. They avoid to write JavaScript code mixed with the HTML code, and they are connected with the view using ng-controller directive.

Service

According to Kumar (2016) a service:

- Provides method to keep data across the lifetime of the angular app
- Provides method to communicate data across the controllers in a consistent way
- This is a singleton object and it gets instantiated only once per application
- It is used to organize and share data and functions across the application

Although users can also build their own services, some built-in Services give additional functionalities to Controllers like:

- Fetching JSON data from a web service with $http
- Logging messages to de JavaScript console with $log
- Filtering an array with $filter

For memory purposes, controllers are instantiated only when they are needed and discarded when they are not. Because of this, every time a page is switched or reloaded, Angular cleans up the current controller. Services however provide a means for keeping data around for the lifetime of an application while they also can be used across different controllers in a consistent manner. (McGinnis 2014.)

AngularJS (2016) provides five recipe types that define how to create objects:

- **Factory**: Is an object with properties that will return the same object. When it is a dependency of a controller, every call the arguments are passed in again, having access to the properties of the factory. Only factories give the possibility to hide private variables. They are the most common used of the three.
- **Service**: Is quite similar to a factory, but with the difference that when it is declared as an injectable argument, it provides the instance of a function passed to module.service (Rocks 2014). In other words, it is a singleton object created by a service factory (Patel 2013).
- **Provider**: Is the only service that can be passed to the .config() function. It is used when module-wide configuration is desired. An important provider is $routeProvider, which helps to divide the application in logical views and bind them to controllers. It implements deep-linking URLs, as shown in Figure 7. The Provider recipe is the core recipe type and all the other ones are just syntactic sugar on it.
- **Value**: It cannot be injected into configurations, but it can be intercepted by decorators.
- **Constant**: The value of a constant should never be changed, it can be injected anywhere but it cannot be intercepted by a decorator.

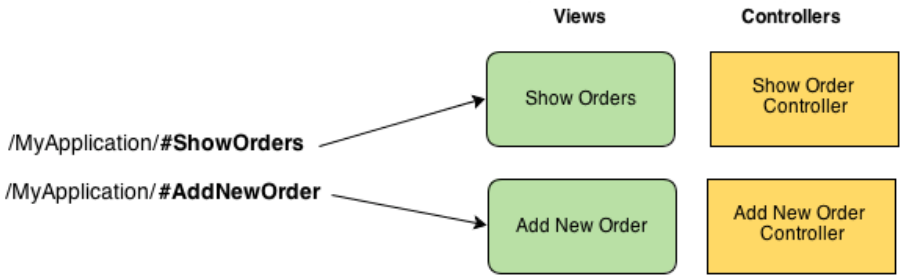

FIGURE 7. Deep-linking (Alejos 2015)

Table 3 shows a visual summary of the main differences that have the five recipes, so that the developer can choose which of them choose for specific implementations.

TABLE 3. Features / Recipe type

	Factory	Service	Value	Constant	Provider
Can have dependencies	Yes	Yes	No	No	Yes
Uses type friendly injection	No	Yes	Yes	Yes	No
Object available in config phase	No	No	No	Yes	Yes
Can create functions	Yes	Yes	Yes	Yes	Yes
Can create primitives	Yes	No	Yes	Yes	Yes

4.1.4 Best practices

Like many other frameworks, AngularJS also has some best practices to follow. It is important to know them because they tell a better and more efficient way of programming, which is always good.

The first one is to implement Immediately-Invoked Function Expressions (IIFE). It is a JavaScript standard to isolate scope of functions. JavaScript has a single execution scope, so as many variables are loaded, there are more probabilities to run into name collisions. IIFE keeps the variables localized to the scope of the function being executed. The function executes immediately, thanks to the parents at the end of the function call. The syntax is:

```
(function () {
    /* code */
}) ();
```

The second good practice is how to **initialize a module**, one way is to declare the module as a variable:

```
var testApp = angular.module ('testApp', []);
```

However this is not the best way to do it, because it will create the module each time, over-writing the existing one (setter syntax). It is preferable to use a getter syntax to retrieve an existing module:

```
(function() {
    angular
    .module('testApp'. []);
}) ();
```

Next good practice is to utilize **named functions** for code readability, as it reduces the amount of callback functions in the code. As an example, the next code has nested lines that make more difficult the readability.

```
(function() {
    angular
        .module('testApp', [])
        .controller('testController', ['$scope', 'testService',
'$log',
            function testController ($scope, testService, $log){
            //Some code
        }]
    )
}) ();
```

In the code below the nesting is avoided, because it creates a long term code maintenance and readability more cumbersome. (Ptacek 2015.)

```
(function() {
    angular
        .module('testApp')
        .controller('testController', testController);

      function testController ($scope, testService, $log){
          //Some code
        }
}) ();
```

Dependency Injection (DI) is related to this code. When minification tools rename the DI functions, they may not be found by Angular. As a result, it is better to explicitly identify DI functions to avoid this. There is a recommended approach using $inject function. (Ptacek 2015.) In the code below is possible to see how it should be used.

```
(function() {
    angular
        .module('testApp')
        .controller('testController', testController);

        testController.$inject = ['$scope', 'testService', '$log']

      function testController ($scope, testService, $log){
          //Some code
        }
}) ();
```

Last good practice is to have a good **folder organization**. One way to do it is separating the code into one folder for Controllers, one for Services, one for Directives, etc. However this method is not the best because in large apps it soon becomes difficult to find the connected code. The best directory structure is dividing the code into functionality folders, for example, if there is a baseball in a sports application, it is much easier to have the controller, service and directive files in a baseball folder. (Ptacek 2015.)

4.1.5 Conclusions

As a conclusion I compare briefly AngularJS with the most popular JavaScript library, jQuery. This library helps to minimize and simplify the JavaScript code, including AJAX support for asynchronous events, and animations to make more visual websites. But, all of this is already included in AngularJS, and it provides more tools for satisfying the current necessities. Angular is not a library, it is a framework, which means that it is also more complex, and it has a more difficult learning curve. But, in order to build large and organized projects, it is worth it. The next step is to talk about Ionic and to introduce the additional features it has above AngularJS, and how it includes Cordova.

4.2 ngCordova

ngCordova is an open-source collection of more than 70 AngularJS extensions on top of the Cordova API that make it easy to build, test, and deploy Cordova mobile apps with AngularJS (ngCordova official website 2015). ngCordova does not call Cordova plugins directly and does not have to figure out the proper object or plugin name, or to check if the plugin is actually installed. It calls a simple AngularJS service, like `$cordovaCamera.getPicture(options)` (Lynch 2014).

4.3 Ionic features

Ionic is an open source front-end SDK for developing hybrid mobile apps with web technologies. It is the front-end UI framework that handles all of the look and feel and UI interactions the app needs in order to be compelling. Since Ionic is an HTML5 framework, it needs a native wrapper like Cordova or PhoneGap in order to run as a native app. (Ionic 2015.)

Ionic provides several features that make the app development even easier:

- Command line utility: makes it easy to start, build, run, and emulate Ionic apps.
- Ionic Lab: build and test iOS and Android versions side-by-side.
- Live reload: instantly update apps with code changes, even when running directly on the device.
- Icon / Splash generation: generates icons and splashes screens for all devices and device sizes with a single command.
- View app: shares Ionic apps with clients, customers and testers all around the world. All without ever going through the App Store.
- Ionic.io Platform: provides full-stack backend services and tools for your Ionic app. A service that embraces mobile web development.
- Ionic icons: offer over 500 custom designed MIT licensed font icons to use with ionic.
- Community forum: allows to ask questions about ionic on their Ionic forum.
- The Ionic book: provides information about Ionic

In Figure 8 it is possible to take a look at the Ionic architecture and have a clearer idea of Ionic. It links AngularJS, Cordova and a powerfull CLI in order to build hybrid mobile

applications. The relation of AngularJS and Cordova with Ionic has been explained above. However, this is the first time that Gulp appears in my thesis. As a brief explanation, Gulp is a built system that allows automating common development tasks, like minifying the JavaScript code and many more. I have decided not to discuss about it, because it is an additional tool for development purposes that I do not use in this project, and it would also need deeper and longer explanation.

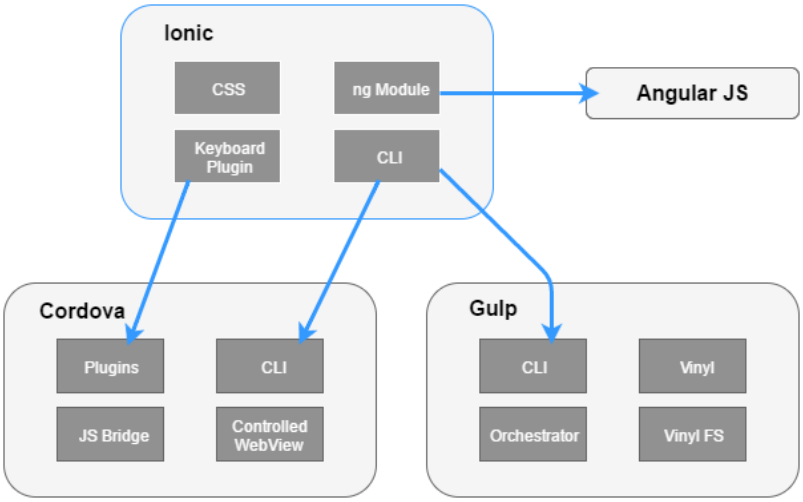

FIGURE 8. Ionic architecture

4.4 Conclusions

In brief, Ionic Framework is a user interface front-end that helps to create hybrid mobile applications. It structures the code application with AngularJS (which implements MVC) and gives the appearance with Ionic directives (created by Ionic with their CSS). It simplifies the hardware access thanks to ngCordova and it provides a powerful CLI to build the app using Cordova and many more improvements that Ionic developers are constantly updating to develop a full hybrid app development tool.

Nowadays, Ionic is also on the way to be a Platform that handles built, push notification, statistics, authorization and deployment processes. The Ionic team is upgrading to Ionic 2 with the update of AngularJS to AngularJS 2, as they have also worked with the Angular team to create the new version.

5 MOBILE UI DESIGN

As Edward de Bono says “Creativity involves breaking out of established patterns in order to look at things in a different way.” This section is dedicated to the appearance side of developing a mobile application, because although many developers look down on this part, it is really important, as it is the link between users and the app. Well-chosen actions may lead to more satisfied users.

Generally speaking, a UI design pattern is a reusable solution to a commonly occurring problem that someone might encounter every day. It is not a feature that can be plugged into product design and it is not a finished design that can simply be coded. Rather, it is a formalized best practice, a guide or template, that designers, developers, and product managers (and anyone else who touches the product) can use to solve common problems when designing a mobile application or system. (Bank 2014.)

In mobile user interface design the developer must pay special attention to the limitations of the mobile device and the device specific control methods. Therefore, the designer must understand the control gestures, input methods and navigation callenges of the mobile devices. In the next sections I will concentrate on these mobile device characteristics in more detail.

5.1 Gestures

Traditional gestures for web interactions have been clicking, hovering and scrolling. Mobile application design has exploded with new design patterns and their implementations. Made possible by advancing hardware and software capabilities, the mobile space is developing with unprecedented levels of human-computer interactions. These solutions are largely empowered by new gestures. And, marked by responsive design advancements, web and mobile design is rapidly converging. As a result, applications may be built for all device shapes and sizes - this will have a dramatic and re-invigorating impact on the design of the web experience. (Bank 2014.)

Some examples of the most common gestures are:

1. Touch
2. Double touch
3. Double touch drag (Figure 9a)
4. Long press
5. Swipe or drag (Figure 9b)
6. Long press drag (Figure 9c)
7. Pinch open & Pinch close (Figure 9d-e)

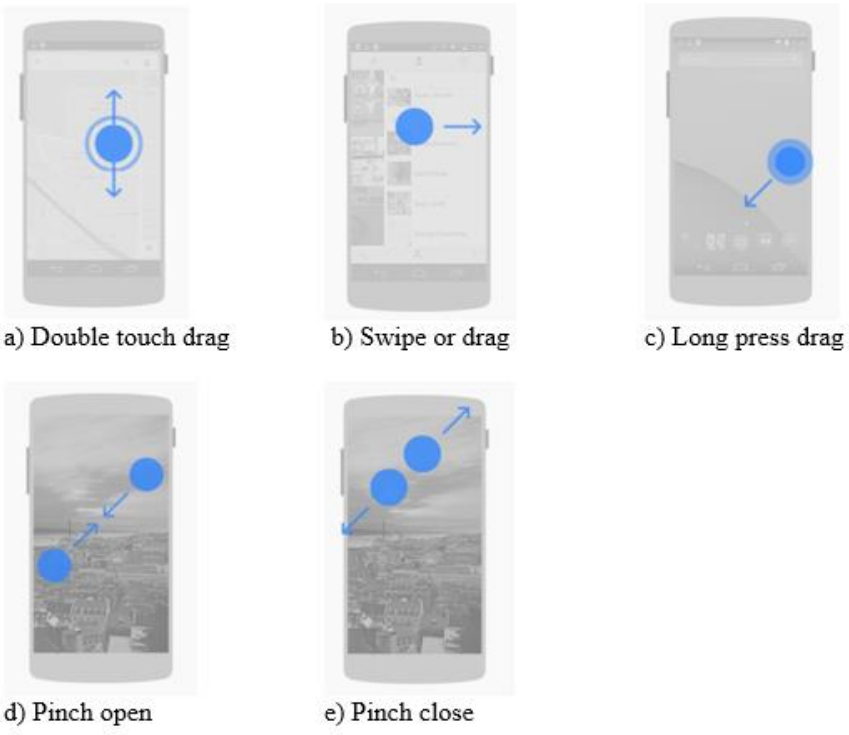

FIGURE 9. Different gestures

Not only does this help preserve screen real-estate by eliminating some of the on-screen buttons, but it also makes the experience intuitive and fun. Combine this with various animations, and we can have a field day with the ways we can implement UI design patterns in mobile applications. (Bank 2014.)

5.2 Getting input

The user interface designer must always bear in mind the limitations of the mobile device input methods. Therefore, it is important to take into account some good practices to create a

beautiful, intuitive and easy to use user interface. Acording to Bank (2014) the main issues are:

1. **Smart keyboards**: Give the user the keyboard that is relevant to the data they are entering in order to enter information quickly. For example, when entering phone numbers in address books or dialers, the user doesn't need the full keyboard.
2. **Default values** and **autocomplete**: Anticipate frequently selected items and make data entry easier for the user by providing them default values. This can be paired with autocomplete functionality, to improve the user experience by speeding things up.
3. **Immediate immersion** or "**lazy signpups**": Some apps can allow their users to come in and use the app before asking them to identify themselves. Oftentimes, registration comes with an added benefit, like cross-device syncing. Late registrations may not always be a good idea, but the option to "try-before-you-register" can be a great way to increase engagement with the app.
4. **Action bars**: Provide quick access to important actions from the app's action bar (or "toolbar" in iOS terminology). While navigation bars have dominated web and early mobile application design, the use of other patterns like drawers, slideouts and sidebars, links to everything, button transformations, vertical and content-based navigation have allowed for more simple app views that can focus on primary and secondary actions, and less on secondary navigation.
5. **Social login**: Integrating social sign in methods that allow users to login through their existing accounts means they have one less username/password combination to worry about.
6. **Huge buttons**: The ideal touch screen tap target size may be 72px, but some apps also give huge buttons so the user knows exactly what to do and can do it quickly wherever he is.
7. **Swiping** for actions: allows content to be swiped or moved out of the way provides users with a very intuitive way of handling the information on screen.
8. **Notifications**: highlight recent activity by visually marking new content.
9. **Discoverable controls**: The user wants quick access to controls that are secondary or only relevant to specific sections or content in the application. Clearing up the clutter and letting users discover particular actions only when they need them. These invisible controls can be accessed by any gesture - swipe, tap, double-tap, long-press, etc. This gives the ability to keep these actions off-screen, saving some valuable real estate.

10. **Expandable inputs**: The user wants to focus on the content instead of sacrificing screen real estate to controls. Designing controls that expand when the user taps on them, keeps most controls out of the way until the user needs them.
11. **Undo**: Provides an easy way for users to undo their actions instead of just asking them to confirm deletions beforehand. Undo prevents situations where an action can cause inconvenience or loss of data if done by accident.

5.3 Navigation

Navigation is also a fundamental part of the UI because it tells the user how to interact with the different parts of the app. As seen in getting inputs, there are also some good practices to follow related to navigation according to Bank (2014):

1. **Walkthroughs** and **Coach Marks**: The user wants to know how to use the different features of the application so it is good to design a walkthrough or tutorial that demonstrates how each function works. There are two basic ways of doing this. Highlighting important parts of the UI with “coach marks” to explain what they do, or using the first launch to show a slideshow that walks users through the entire experience, effectively explaining what the user can accomplish with the app.
2. **Overflow menus**: Hide extra options and buttons in an overflow menu so that they do not clutter the main interface.
3. **Sliders**: In order to move between options it is a good practice to make transitions between selections and easy with the swipe of a finger.
4. **Content-Based navigation**: Make transitions between overview and detail states seamless. This creates an extremely fluid and intuitive user experience and flow.
5. **Morphing controls**: Replace buttons and on-screen controls with alternative functionality, depending on what the user is currently doing. For example, transforming the ”+” into an ”x”button.
6. ”**Sticky” fixed navigation**: Users want to have access to the menus anytime while in the application, so the top, side or bottom navigation stays in place while a page is scrolled.
7. **Vertical navigation**: When users need a way to navigate between different sections of the app, but there is limited space to show this information important sections of the UI are presented in a list, which the user can scroll through to get what they want.

8. **Popovers**: If the user wants to view relevant information without losing their current place in the UI it is a good solution to show this notifications in popovers. This UI pattern has the advantage of providing a lightweight and straightforward way of viewing information or taking an action without pulling the user out of their current activity.
9. **Slideouts**, **sidebars** and **drawers**: When the user needs to navigate between different sections of the app without being distracted in each individual section it is possible to create a secondaty section of the application in a collapsible panel, and hide it when it is not needed (Figure 10).

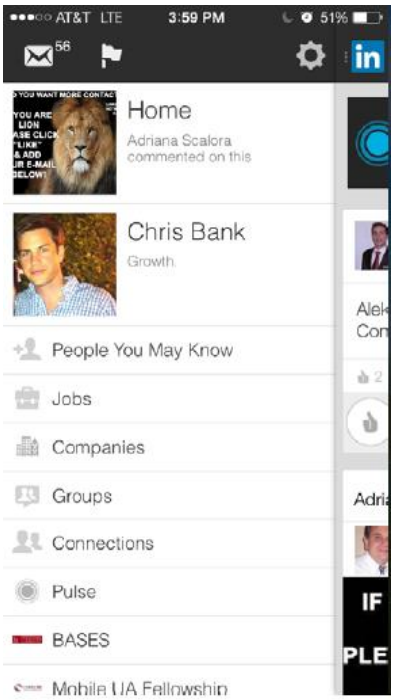

FIGURE 10. Sidebar

10. **Links** to everything: Users need a consistent way of navigating through content without being distracted by additional content. If they want to interact with a piece of content in the app, odds are that they can tap on it and go to a new view for a more detailed experience.
11. **Advanced scrollbars**: Users need to see their current position in the context of an entire content set. Beyond scrolling with a swipe gesture, mobile lists and galleries have a persistent or temporarily scroll bar, and it can be a scroll index – dates, alphabetical letters, categories, etc.

12. **Swipe views**: Allows users to move from item to item by swiping through content without having to go back to the index. This pattern should be familiar from browsing through photo albums, but more and more apps are starting to implement this for their content as well.

5.4 Data and content management

It is crucial to show the information properly and clearly, so that users do not have difficulties to read it. Coming up next are the best patterns to apply when showing content according to Bank (2014):

1. **Full-Screen modes**: Users want to focus on content instead of being distracted with the UI hiding or minimizing the UI around content, helping to focus on what really matters.
2. **Inline expanding areas**: Make metadata invisible unless users explicity want to see it, for example, hiding individual timestamps on chats.
3. **Transparency**: Uses gradients and fading overlays to show that there is content layered below (Figure 11).

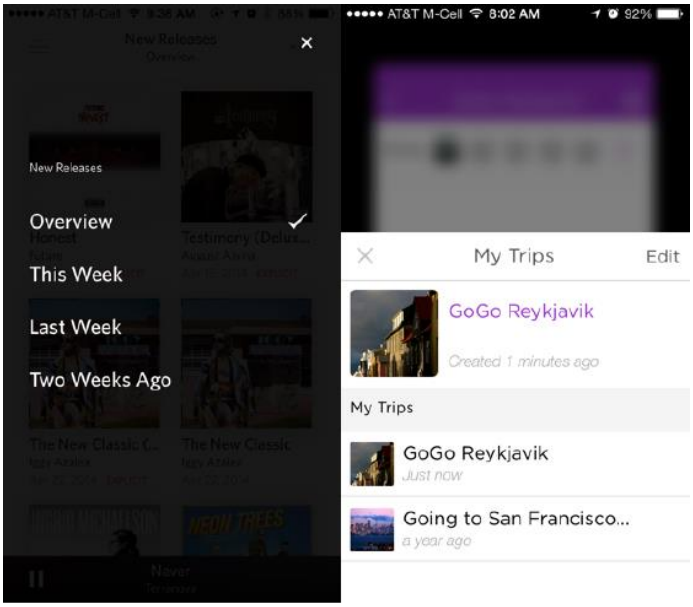

FIGURE 11. Transparencies

4. **Grids**: Shows snoppets of content in a grid so that it is organized.
5. **Cards**: Provides browsing through content quickly and interact with it, present snippets of information in bite-sized cards that can be manipulated to show more information if the user wants it.

6. **Empty states**: The user also needs to know what section of the application is empty and what to do next. Making sure the UI provides a good first impression by designing for the ”blank state” is the condition when there is no user data (Figure 12).

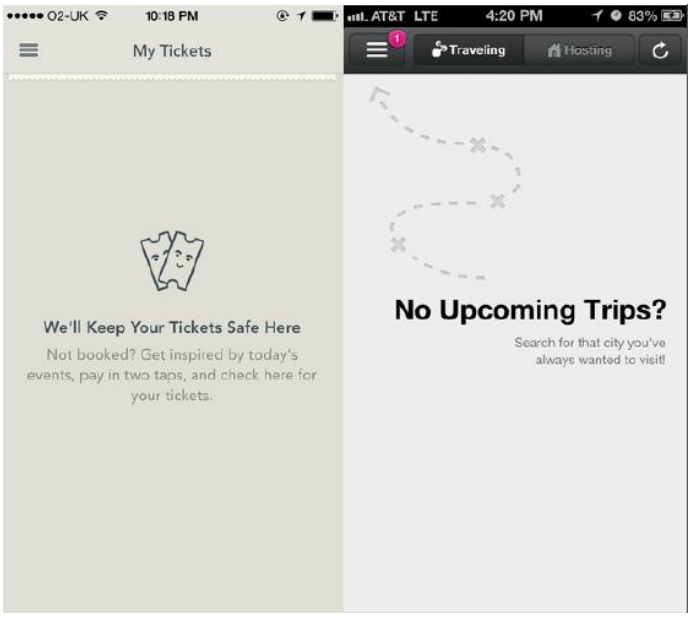

FIGURE 12. Empty states

7. **Direct manipulation** of content and data: Allows for content to be edited directly without having to transition between editing or deleting modes.
8. **Draggable objects**: In order to sort and organize items in a way that makes sense to users in the current view, allowing to move items around, pressing-and-holding then dragging-and-dropping them wherever the user wants.
9. **Pull to refresh**: Users also want to be able to refresh the content manually.

6 APP IMPLEMENTATION

In this section I will cover all the hybrid mobile app development process I made, from the client side to the server side. I will start from the Ionic installation process, followed with more detailed implementation explanations.

My app called What’s for lunch (What’s 4 lunch?! on the .apk) contains a list of recipes which users can consult, and filter according to two categories I have implemented. The first one is called Search and it filters the recipes based on some level of difficulty, food category and

preparation time. The second filter is called My kitchen and it can search recipes based on the available ingredients users have, through a list of recipes that s/he can cook with them. Finally, there is a Random section called Astonish me! that outputs a random recipe.

6.1 Architecture

The app uses the client-server architecture which allows focusing on the client side, the Ionic app, the visual part and the filters operations, and the model, data and image storage on a server side, Backand (Back&) and Google App Engine (GAE). Figure 13 shows how the app connects with servers, and it clearly highlights the client and server sides. The Ionic app is on the client side, using phonegap to communicate with the device and Ionic and AngularJS for the front-end. The app gets JSON data through an API REST on the Back& server, and the images loading GAE URLs. Both parts will be explained with more details in Sections 6.2 and 6.3.

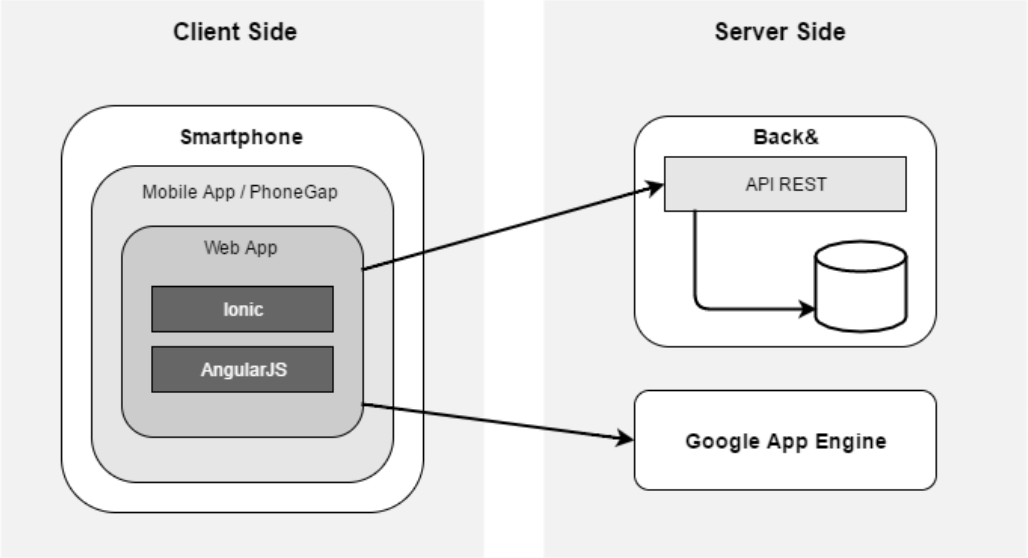

FIGURE 13. Client-server architecture

6.2 Client side

In this subsection I will explain the Ionic installation process necessary to develop this app and all the details related to the app. The User Interface, controllers and performance issues I have faced are also an important part of this client side explanation.

Installation

Before I am able to create my Ionic app I must configure the working development environment. I will go through the installation process step by step in a four-step process tested on Windows 7 and Windows 10 Operating Systems.

Step 1: Install Cordova dependencies

First, the Cordova dependencies must be installed, following the steps from the Android Platform Guide documentation on Cordova website. Download and install:

1. Java jdk
2. Apache ANT
3. Android sdk
4. Git (from http://git-scm.com/downloads)

Configure environment variables

- PATH = bin folder of java; Platform tools folder of Android sdk; Tools folder of Android sdk; Bin folder of Apache ant
- JAVA_HOME = jdk folder of java

Step 2: Set up Android SDK

Open Android SDK manager typing ’android’ in the cmd and install:

- Tools – Android SDK Tools
- Tools – Android SDK Platform-tools
- Tools – Android SDK Build-tools
- API 19
- Extras – Android support library
- Extras – Google USB driver
- Extras – Local Maven repository for Support Libraries

Step 3: Install Node JS

1. Download and install Node JS from https://nodejs.org/.
2. Add Nodejs folder path to the environment variable PATH.
3. Open the cmd and type ‘node’ and see that it is installed.
4. Open the cmd and type ’npm’ and check that it is installed.

Step 4: Install Cordova & Ionic

Open the cmd and type:

1. npm install –g cordova
2. npm install –g ionic

By typing 'ionic' in the cmd it is posible to see that it is installed as shown in figure 14.

FIGURE 14. Ionic installed

Visual appearance

Now that the client-required environment is correctly installed, I will show how the app looks like. The first detail users see when they install an app is the Icon (Figure 15). What's for lunch icon has the app colors: red, orange and yellow, and a simple spaguetti bowl. It has been designed using Adobe Illustrator to create a simple vectorial white pattern and Adobe Photoshop to create the background following the new Material Design icon styles, using a rounded shape.

FIGURE 15. What's 4 Lunch?! icon

The second detail users see is the Splash Screen, when the app is launched. I decided to skip this, because I personally think it must be used only on heavy apps that have a relative long loading time, and what I want is to give the feeling of a fluid, simple and fast app. Because of this my Splash Screen is white and I have modifyed the delay time to zero in 'config.xml':

```
<preference name="SplashScreenDelay" value="0"/>
```

The initial view when the app appears for the first time is the Home screen ('home.html'). As shown in Figure 16, it has a common app element, the red header bar with the app name, and three big sections to select the desired action. This choice has been made aiming to cover all

the screen. As it is a simple app, it always concentrates the user attention on what really matters and tries to simplify the interaction. Another important applied element on this view is the text, as it is more readable thanks to a slight darkening and blurring of background images. These are the only decorative elements, and also the text shadow to emphasize it.

FIGURE 16. Home view

Search and My kitchen buttons display a list of recipes (’search.html’ and ’kitchen.html’) using Grids and Cards to arrange them. Figures 17 and 19 show how each recipe is previewed with an image and a brief description. There is also a red header with Home and Filter/Cart buttons. Because the name of the recipes is displayed on the bottom of the images, I have used a soft shadow to give visilility to the white text. Figures 18 and 20 show Search and Kitchen filters (modal pages), and in this case there is a common button to search on the footer.

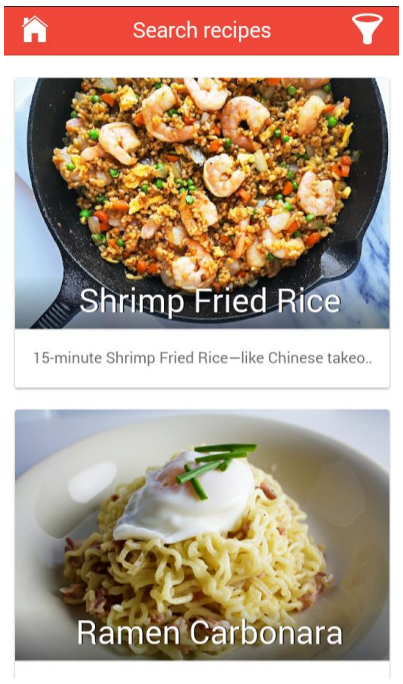

FIGURE 17. Search view

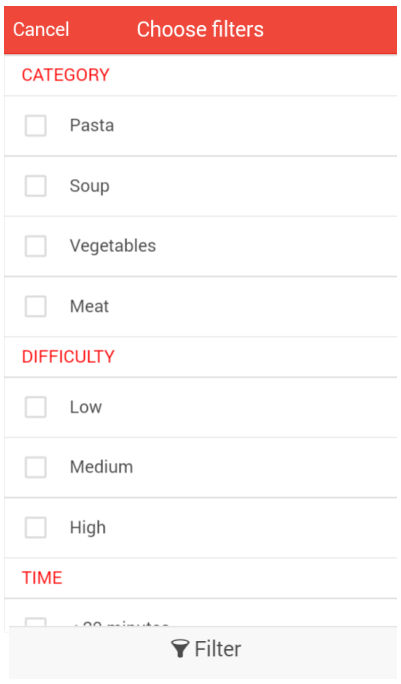

FIGURE 18. Search filter

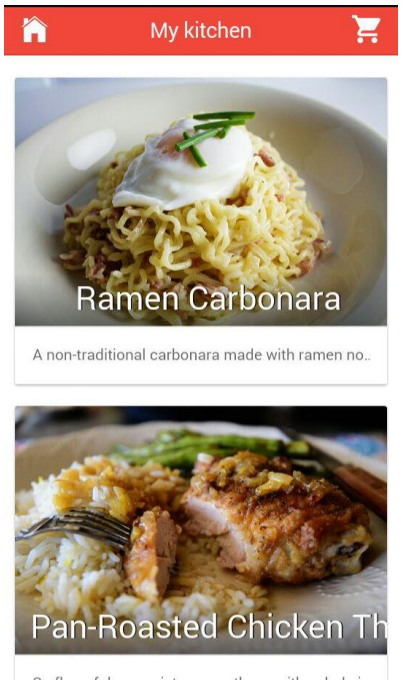

FIGURE 19. Kitchen view

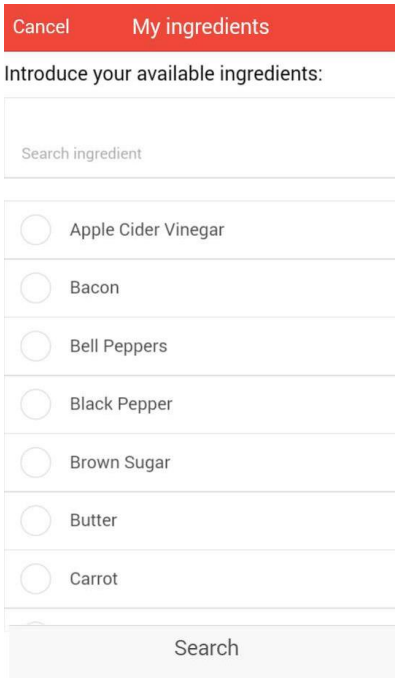

FIGURE 20. Kitchen filter

Pressing a recipe image and also the Astonish me! button, the recipe details (’recipeDetails.html’ and ’recipeDetailsRandom.html’) are displayed. As Figures 21 and 22 introduce, both show the recipe image followed by its details, and the separators Ingredients and Instructions have the app red color to give a continuity to the app appearance. The Recipe details view has only one back button inside the header bar, while the Random recipe details

view contains a home button on the left, and a swift button on the right to allow the user to display another random recipe.

Last but not least, as explained in Chapter 5, an important UI element is empty states that tell the user why there is no data. ”What’s 4 lunch?!” app shows a simple text message when the filter parameters do not match with any recipe, as shown in Figures 23 and 24.

To sum up, all the app has common elements and colors, the icons are simple and white to be clearly distinguishable, and there is always the same structure to go back and forward on all the related views. In this way the app user interface is clear and easy to use.

FIGURE 21. Recipe details view

FIGURE 22. Recipe details Random view

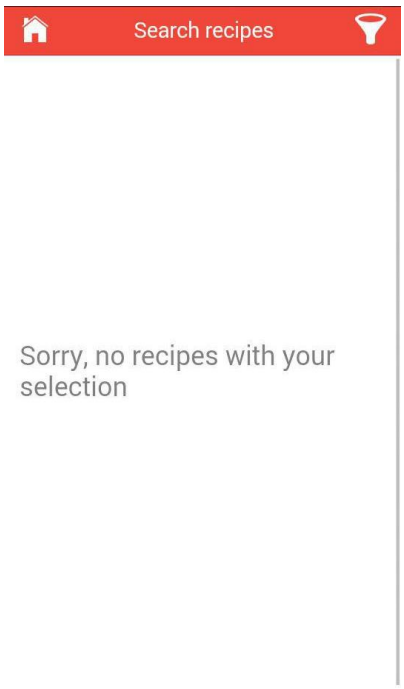

FIGURE 23. Empty Search message **FIGURE 24. Empty Kitchen message**

Code structure

I have the code separated into files by their functionality, so that I have three main functions called: search, kitchen and random. Search functionality is related to the filter by food categories, difficulty levels and cooking time. Kitchen functionality is related to the filter by ingredients that shows the recipes that can be made with them. Random functionality is about selecting a random recipe and show its details.

Almost every view is controlled by a Controller, which connects with the model and provides the data to be displayed. Home view is the only one that has no controller because it does not need any variable information, and it is the default view as configured on 'app.js'. Views and controllers are linked in 'app.j's through $stateProvider and making use of some Nested States. Nested States are a way of creating child states, heritage, using a dot notation, for example 'app' is the ancestor of 'app.search', so when 'app.search' is active, so does 'app', thus search view can also use 'app''s controller.

Figure 25 shows the app files, standing out templates (views), controllers and factories (database connection). Index.html is the main file. It is the one that, through `ng-app="w4l"`, calls the main module w4l which loads AngularJS routing connecting the views with their controllers. Index.html is the main template, where the rest of templates are displayed having

access to the css files. Factory files are also loaded on this file, but are only injected on the controllers that need them, as indicated with the arrows. Double arrows indicate the two-way data binding between views and controllers.

'App' state is an Abstract State, which means that it can have child states but cannot get activated itself. In consequence, menuContent.html is not a separated view, as indicated on Figure 25. Underscore library is also loaded on 'index.html', and injected to w4l module so that it can be used throughout the app.

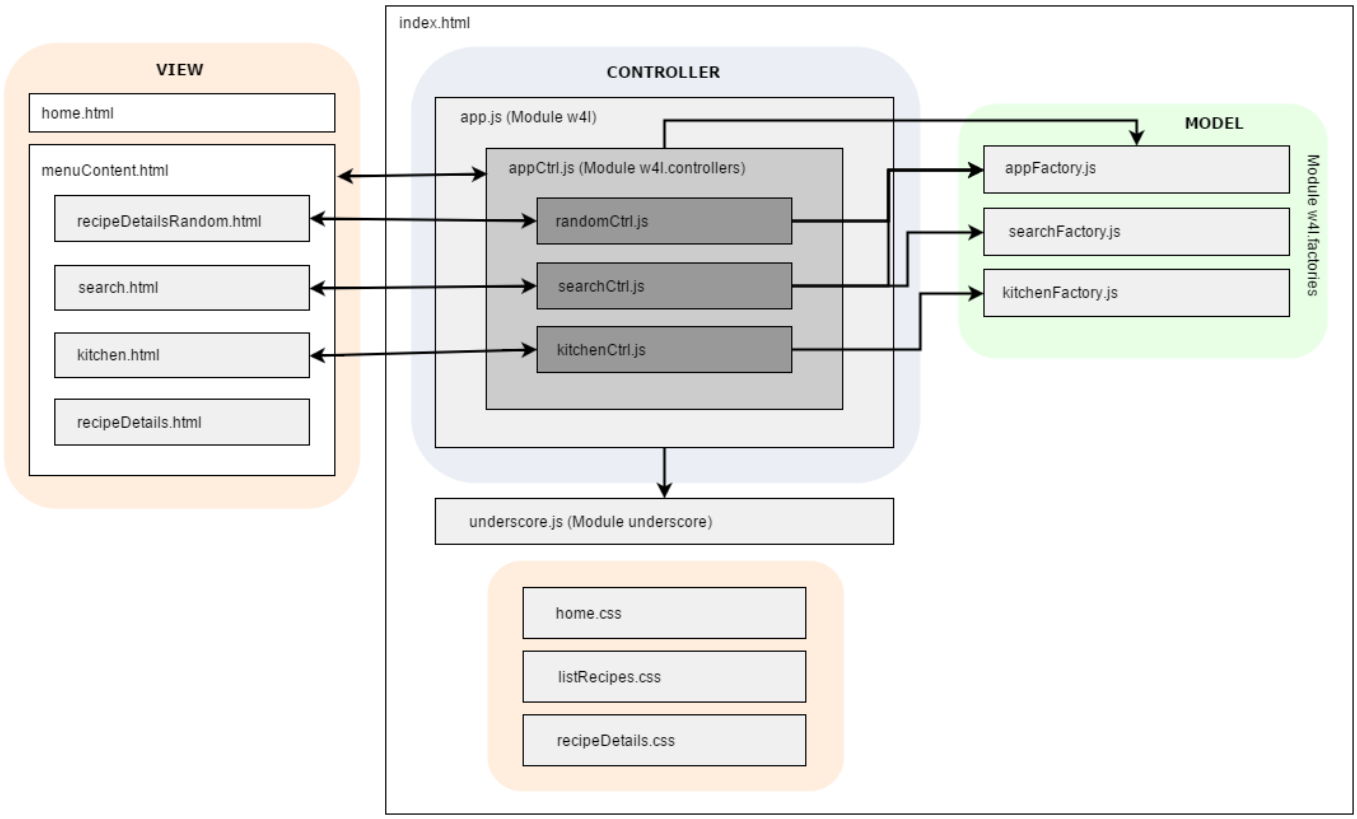

FIGURE 25. App file structure

Performance issues

Crosswalk project boosts Cordova apps running on Android. If the app has an old Ionic architecture, Crosswalk is not installed by default. The problem comes from Android Chrome Webview, which its Javascript engine, on Android versions anterior to 4.4, has really poor performance. This means that the app runs very slowly on these Android versions, and there are also a lot of inconsistencies between devices when running an app. Developers had to do some specific css fixes for every version.

Crosswalk project was created to solve this problem as it is a bundled Chrome webview that replaces the default Android device Chrome webview. Therefore, every device running the app have the exact same webview. No more performance problems and no more css fixes for older versions. Crosswalk has one downside, since it increases the app size about 15-20 Mb, but it is worth the size due to the huge performance improvement and certainty to render the app perfectly on every Android version. The Ionic team made a really easy installation process. Using Ionic CLI it is installed with the command `ionic browser add crosswalk`. (Mitch 2015.)

On touch devices, such as a phone or tablet, browsers implement a **300ms delay** between the time the user stops touching the display and the moment the browser executes the click. It was initially introduced so that the browser can tell if the user wants to double-tap to zoom in the webpage. Basically, the browser waits roughly 300ms to see if the user is double-tapping, or just tapping on the screen. While 300ms seems pretty short, it is surprising just how noticeable it is when the delay is removed. (Bradley 2014.)

Ionic has made a great job implementing their own solution. Although they have consider Angular's ngTouch and FastClick.js, they explain in their blog the reasons why they have not used them, and how they finally have solved this problem that caused an extra delay on hybrid apps.

Ionic uses **javascript scrolling** by default, which tends to get choppy. If there are a lot of images there are major issues, especially on Android. As stated in their official Ionic blog, this has some issues on iOS. They seem to have fixed most of the bugs at least on iOS9 and everything works including pull to refresh and infinite scroll (Bolinger 2015). To fix this on Android, I have enabled native scrolling on my app. It can be activated globally adding to the 'app.js' the following lines:

```
.config(function($ionicConfigProvider) {
   if(!ionic.Platform.isIOS())
      $ionicConfigProvider.scrolling.jsScrolling(false);
});
```

By default, there is already a caching mechanism present in Ionic. However, this mechanism only caches the view when it is first entered. By using **Angular TemplateCache** it is possible to make sure all views are already cached at app startup. This means no more initial lag when

entering views for the first time (Buyse 2015). I have considered that this performance improvement is not convenient for my app, because if the list of recipes were too large it would require too much resources to be all preloaded. That is why I think this improvement it is better to be used on simple and small apps.

Another improvement realted to caching is **Image Cache**, through $ImageCacheFactory. This solution is more selective and I consider that, on apps like mine, it can be useful to preload the images that are going to be immediately loaded, hence improving the user experience without image waitings. This solution is similar to **lazy loading**, useful when dealing with remote images. Lazy loaging is a design pattern that consists on loading the required data just when needed.

Bigger apps would need more research and I consider John Papa (2016) github repository a good starting point, reading the **Angular style guide** to get a better idea of how to correctly use AngularJS.

6.3 Server side

In this subsection I will go through the server side development process. I will explain the tools I have used for this part, and how to connect them to my app. But, first of all, I will talk about the possible alternatives I have consider before making a decision.

The first idea I had about the implementation of the server side was to use a food API, so that I could have access to many recipes without worrying about the server and just learning how to connect to the API in my app. I found out among others: Yummly Recipe API, BigOven API, Food2Fork and FatSecret Platform API. The reasons why I discarded this option was that many of them were paying APIs, if they had a free plan, it was too limited and I was not sure if it was convenient to adapt all my app to show the information the API provides me. This means that my app requires specific information about each recipe, like the preparation time, food category or difficulty preparation level, that none of the APIs had.

The second idea was to create my own RESTful API server using Google App Engine, as it offers free server service managed by the user. I started with this option using GAE combined with Bottle, a micro web-framework for Python to create a RESTful service and the basic

Create, Read, Update and Delete (CRUD) operations. After many failed attempts to send JSON data to my app, I just got to work images.

Finally, I decided to search for a backend service adapted to the Ionic framework, capable of sending JSON data and easy to connect on hybrid apps. I found out Firebase, Parse and Backand. The first one seemed to be a really nice option, but it had a very limited free plan with only 100 connections forever, and just in development procces I would need more. The second one, Parse, also had nice comments and documentation but they have announced that they will retire hosted services on 2017, so I prefered to use another one that was not going to end the services. The last one, Backand, is a new service defining themselves as ’backed-as-a-service for AngularJS’. They had easy examples using Back& with Ionic and Ionic 2, the registration before May was totally free (free-forever account) and they also had a nice web interface that allows to create a JSON database model. These reasons gave me a clear choice, **Back&**.

After the registration process, the first thing I made was to create a New App called whats4lunch and to configure the model. Figure 26 shows the JSON app model, with relations one-to-many and many-to-many. Next step was to save the recipes data into this database with a JSON scheme, for example, table ‘level’ is fill out with the next JSON code:

```
[
  {"name": "Easy"},
  {"name": "Intermediate"},
  {"name": "High"}
]
```

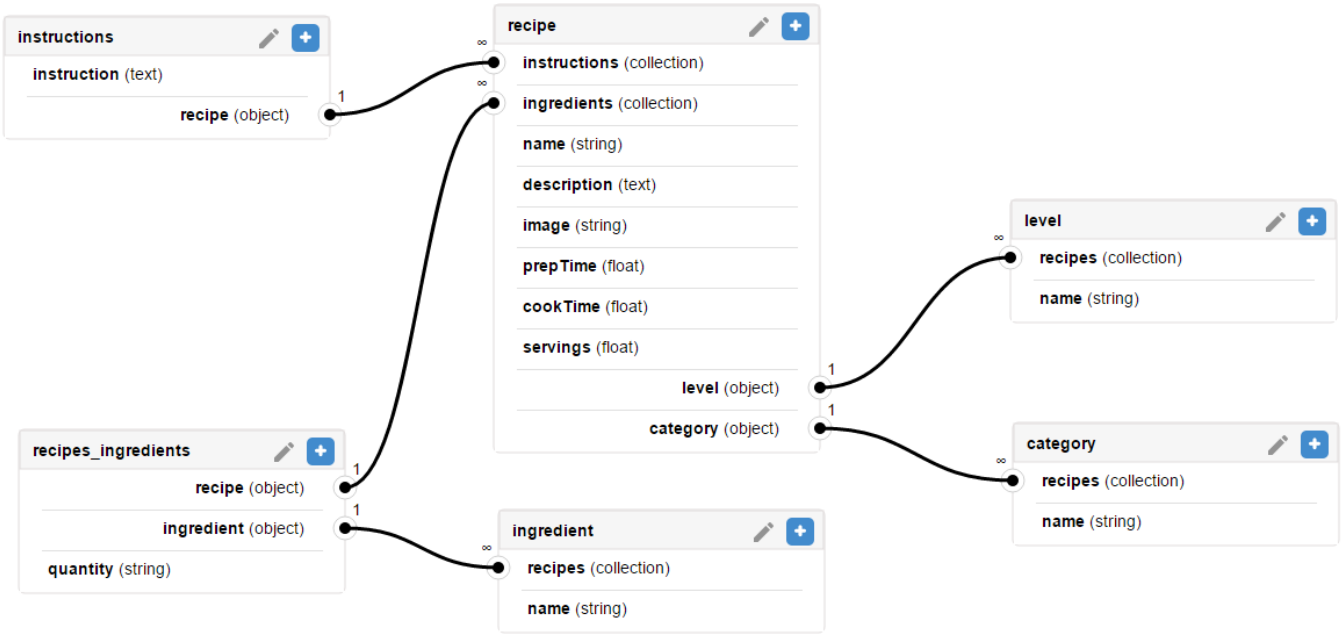

FIGURE 26. Backand model

After this, the Back& was configured and all the recipes' data on it, but there were no images. Backand also offers a storage file service, but as I had up and running the images on Google App Engine I considered easier in this case to continue using it. Server side was fully configured and the only thing to be done was the app integration.

App integration

Back& integration was really easy as they provide good documentation and it is done following some simple steps. However, in my case it was necessary to previously install a Whitelist plugin. The new Ionic versions already include this plugin on the starter app, but it was not my case and I run into problems because of this. Whitelist Cordova Plugin is necessary to make network requests, otherwise requests return http 404 errors. In order to install it, it is made through Ionic CLI with the below command and adding the following lines to config.xml file to allow requests:

```
ionic plugin add https://github.com/apache/cordova-plugin-
whitelist.git
<allow-navigation href="http://w4l-recipes.appspot.com/*" />
<allow-navigation href="https://api.backand.com/1/objects/*" />
```

Back& service is integrated in the app following a three-step process:

1. Include the following code in index.html file above the project's code:

```
<!-- Backand SDK for Angular -->
<script
src="//cdn.backand.net/backand/dist/1.8.2/backand.min.js"></script>
```

2. Add 'backand' to the module dependencies of the angular app in app.js:

```
angular.module('w4l', ['backand'])
```

3. Backand uses OAuth2 authentication, which requires including the authentication token in every HTTP call:

```
myApp.config(function (BackandProvider) {
    BackandProvider.setAppName('App-Name');
    BackandProvider.setSignUpToken('SignUp-Token');
    BackandProvider.setAnonymousToken('Anonymous-Token');
})
```

Following these steps the Ionic app can access Backand's API endpoints using $http requests. The best way I have seen to get the data is using a Service or a Factory. Backand documentation uses Services, however a service return a method and what I want is to have an Angular component that gets and returns the JSON data. For this reason, I have implemented Factories. The following code is a brief example of how my app access the data. In this case it gets the JSON data from recipe's table:

```
angular
.module('w4l.factories', [])
.factory('appFactory', function($http, Backand) {
    var baseUrl = '/1/objects/';
    function getUrl(objectName) {
        return Backand.getApiUrl() + baseUrl + objectName;
    }
    function allRecipes(){
        return $http.get(getUrl('recipe/')).then(function(response) {
            return response;
        });
    };
    return {
        all: function() {
            return allRecipes();
        }
```

```
    };
});
```

Controllers that inject appFactory can access to the method ’all()’ which calls to local ’allRecipes()’ method which has the $http request. Because an $http.get call is asynchronous, data must be accessed on an asynchronous way, using a **Promise-Based architecture**. Controllers that need to manage data must implement this asynchronous architecture in order to work with it.

The following paragraphs are an explanation of this Promise-Based architecture, as it is a key concept using external APIs like Backand, and it is mainly based on McGivey (2015) website. In synchronous architecture, the next set of instruction cannot be run until the last set has completed finish. In an app, this can cause the app to appear non responsive as it tried to complete tasks such as getting data from an external API.

A promise is a guarantee that you will receive something in the future. Going with that assumption, the app plans what to do when that promise is fulfilled (technically known as resolved). In other words, when it asks for something, instead of getting a response right away, it gets a deferred response and it defines what to do when it finally gets the actual response. For example, from a controller, my app wants a list of ingredients from Back& API. Traditionally, it would look like this:

```
$scope.ingredients = IngredientsFactory.getAll();
```

As stated above, because this is synchronous, the code execution must stop until it gets a response back. Using promises, it will call the same method, but this time, it will say something along the lines of “Get all ingredients, then store them in the ingredients variable.”

```
IngredientsFactory.getAll().then(function(data){
    $scope.ingredients = data;
});
```

To summarize, the controller requests something, is given a promise that it will get a response, and when it does get that response, it handles it in a then. Finally, the GAE images

integration is done simply using `<img src="http://w4l-recipes.appspot.com/img/{{recipeDetails.details.image}}">` HTML tag.

7 CONCLUSION

Nowadays Android and iOS are the two main mobile operating systems, and Windows 10 is strongly committed to improve their OS for touch devices. This diversity leads to different user experiences depending on the OS they use, so the idea of hybrid apps gets stronger and, meanwhile, performance issues are also being improved to be able to compete with native apps. Many different hybrid solutions already exist, started with PhoneGap project, which every year attracts to more developers, creating competitors to native apps like Ionic Framework.

My experience in developing and researching during this thesis is more than positive, because I have learned new technologies, frameworks, architectures and I have gained knowledge regarding to state-of-the-art in hybrid mobile applications. I started this thesis in 2015 and I finished it in 2016, one year later, so I could also experience how much this area of IT is changing, day by day. In the beginning of 2015 Ionic 1 was released and I was excited about this new mobile app approach. Only one year later AngularJS 2 and Ionic 2 have been launched, with new features and improvements. This fact only demonstrates and strengthens the idea that, although smartphones are really new devices, they evolve quickly creating new and better solutions to the modern life style.

In the theory part, I have explained the main concepts of Ionic, how it is related to the Cordova project, how important AngularJS is, and some basic mobile user interface guide lines to create good quality mobile applications. Now that I have a clearer idea, I see that Ionic can be a good option for apps that do not require really high performance, specially on Android devices. I have tested my app on different Android versions and I have found performance issues under Android 4.4, and also that iOS behaves better. Hence, in the practical part I have solved implementation problems, I have learned how to improve performance and I also have understood and applied the theory concepts I have documented. Even though 'What's 4 lunch?!' is not a big app, I consider that it is big enough for testing this framework, having also the complexity of creating a server side infraestructure.

From my point of view, having like endorsement this thesis, the hybrid solution is the result of the desire of having an easier way of developing mobile apps. There is such a OS division that the real objective is blurred, create good and useful apps for everyone. Developers have to create different implementations depending on the mobile OS, the OS and browser version and the manufacturer device, which only complicates and gives a worse app end result. In my opinion good performance is an important requirement that must be met. Because of that, I have searched all the possible solutions to have better performance on my app and I found that there are already some of them. However, hybrid web-based apps depend on the WebView layer, which on Android was not unified until version 5.0, and this can still be improved. Moreover, these solutions will always add this additional layer, the WebView, which will be always an extra overhead.

On the other hand, I have talked in this thesis about cross-platform solutions which, in pursuit of the same idea, reuse most part of code to create, in this case, native apps. I consider this as an alternative option to be studied and tested, in order to check if it works as they claim. Lastly, I want to end up saying that, taking into account the hybrid app advantages and disadvantages, Ionic Framework is one of the best options thanks to its structure, developer team, community and support.

BIBLIOGRAPHY

AngularJS 2016. Providers. WWW document. https://docs.angularjs.org/guide/providers. No update information. Referred 01.05.2016

Bank, Chris 2014. Mobile UI Design Patterns. eBook. http://studio.uxpin.com/ebooks/mobile-design-patterns/. No update information. Referred 16.11.2015

Bolinger, Scott 2015. 4 ways to make your Ionic app feel native. WWW article. http://scottbolinger.com/4-ways-to-make-your-ionic-app-feel-native/. Updated 13.10.2015. Referred 08.05.2016

Bradley, Adam 2014. Hybrid apps and the curse of the 300ms delay. WWW article. http://blog.ionic.io/hybrid-apps-and-the-curse-of-the-300ms-delay/. Updated 30.01.2014. Referred 08.05.2016

Bristowe, John 2015.What is a Hybrid Mobile App? WWW article. http://developer.telerik.com/featured/what-is-a-hybrid-mobile-app/. Updated 25.03.2015. Referred 07.06.2015

Buyse, Tom 2015. Improving the performance of your Ionic application. WWW article. http://tombuyse.com/improving-the-performance-of-your-ionic-application/. Updated 18.08.2015. Referred 08.05.2016

Chester, Brandom 2014. Android Lollipop. WWW article. http://www.anandtech.com/show/8207/google-reveals-details-about-android-l-at-google-io. Updated 25.06.2014. Referred 10.10.2015

Clarck, John F. 2015. History of mobile applications. PDF file. http://www.uky.edu/~jclark/mas490apps/History%20of%20Mobile%20Apps.pdf. No update information. Referred 01.06.2015

Crackberry 2015. Blackberry. WWW page. http://crackberry.com/category/blackberry-os. No update information. Referred 01.06.2015

Developer mozilla 2015. Firefox OS architecture. WWW document. https://developer.mozilla.org/en-US/Firefox_OS/Platform/Architecture. Updated 30.09.2015. Referred 17.10.2015

Eadicicco, Lisa 2013. Sailfish OS. WWW article. http://blog.laptopmag.com/sailfish-os-5-things-jolla. Updated 20.05.2013. Referred 05.06.2015

Gahran, Amy 2011. What's a mobile app. WWW article. http://www.contentious.com/2011/03/02/whats-a-mobile-app/. Updated 02.03.2011. Referred 05.06.2015

Gaić, Dragan 2014. What's the best UI framework Onsen UI or Ionic?. WWW forum post. http://www.quora.com/Whats-the-best-UI-framework-of-Onsen-UI-or-Ionic. Updated 5.10.2014. Referred 11.06.2015

Hurtado, Pedro & Jorge, Xavier 2014. Angular JS Manual. WWW manual. http://www.desarrolloweb.com/manuales/manual-angularjs.html. No update information. Referred 13.06.2015

Ionic 2015. All about ionic. WWW document. http://ionicframework.com/docs/guide/preface.html. No update information. Referred 16.11.2015

Ismash, 2015. Evolution of the Mobile Phone. WWW article. https://www.ismash.com/blog/evolution-of-the-mobile-phone. No update information. Referred 01.06.2015

Kumar, Dhananjay 2016. Services in AngularJS simplified with examples. WWW article. https://www.airpair.com/javascript/posts/services-in-angularjs-simplified-with-examples. No update information. Referred 01.05.2016

Lamb, Daniel 2015. A Comparison and Migration Walkthrough. WWW article. https://www.airpair.com/angularjs/posts/jquery-angularjs-comparison-migration-walkthrough. Updated 2014. Referred. 13.06.2015

LeRoux, Brian 2012. Phonegap, Cordova, and what's in a name?. WWW article. http://phonegap.com/blog/2012/03/19/phonegap-cordova-and-whate28099s-in-a-name/. Updated 19.03.2012. Referred 10.06.2015

Lynch, Max 2014. The official Ionic Blog. WWW article. http://blog.ionic.io/ng-cordova/. Updated 3.06.2014. Referred 16.11.2015

Manishankar 2014. What is iOS and how it works. WWW page. http://www.makemegeek.com/what-is-ios-how-it-works/. Updated 06.04.2014. Referred 01.06.2015

McGinnis, Tyler 2014. AngularJS: Factory vs Service vs Provider. WWW article. http://tylermcginnis.com/angularjs-factory-vs-service-vs-provider/. Updated 4.05.2014. Referred 14.06.2015

McGivey, Andrew 2015. Promise-Based Architecture in an Ionic App. WWW article. http://mcgivery.com/promise-based-architecture-in-an-ionic-app/. Updated 13.04.2015. Referred 08.05.2016

Mitch 2015. Boost your Cordova app performance on android with crosswalk. WWW article. http://geeklearning.io/boost-your-ionic-app-performance/. No update information. Referred 08.05.2016

Natividad Alejos, Luis F. 2015. WWW article. http://frontendlabs.io/2152--hablemos-de-angularjs. Updated 18.12.2014. Referred 01.06.2015

Niagaraax 2015. What is a software framework. WWW document. http://www.niagaraax.com/cs/products_/_services/frameworks. No update information. Referred 10.06.2015

Noman Ali, Syed 2013. Types of apps. WWW article. http://www.socialhunt.net/blog/types-of-mobile-app/. Updated 08.10.2013. Referred 06.06.2015

O'Hagan, Tony 2014. Difference between Phonegap and Sencha Touch. WWW forum answer. http://stackoverflow.com/questions/14643527/difference-between-phonegap-and-sencha-touch. Updated 28.05.2015. Referred 10.06.2015

Papa, John 2016. Angular Style guide. WWW document. https://github.com/johnpapa/angular-styleguide/blob/master/a1/README.md. Updated 27.04.2016. Referred 08.05.2016

Patel, Viral 2013. AngularJS routing and views. WWW article. http://viralpatel.net/blogs/angularjs-routing-and-views-tutorial-with-example/. Updated 15.10.2013. Referred 14.06.2015

Ptacek, John 2015. AngularJS best practices. WWW pages. http://jptacek.com/2015/02/angularJS-Best-Practices/. Updated 18.02.2015. Referred 14.06.2015

Quirksmode 2015. Web vs native. WWW article. http://www.quirksmode.org/blog/archives/2015/05/web_vs_native_l.html. No update information. Referred 07.06.2015

Roks, Stefan 2014. AngularJS: Factory vs Service vs Provider. WWW article. http://www.theroks.com/angularjs-factory-vs-service-vs-provider/. Updated 27.02.2014. Referred 14.06.2015

Rouse, Margaret 2013. Native app definition. WWW article. http://searchsoftwarequality.techtarget.com/definition/native-application-native-app. Updated 01.02.2013. Referred 06.06.2015

Sundqvist, Sara 2013.Why native apps are better. WWW article. http://twotoasters.com/why-native-apps-are-better-than-mobile-web/. Updated 05.06.2013. Referred 06.06.2015

Ubuntu site 2015. Ubuntu Touch. WWW pages. http://www.ubuntu.com/phone/developers. No update information. Referred 17.10.2015

Whinnery, Kevin 2012. Comparing Titanium and Phonegap. WWW article. http://www.appcelerator.com/blog/2012/05/comparing-titanium-and-phonegap/. Updated 12.05.2012. Referred 10.06.2015

Wikipedia 2015a. iOS. WWW document. http://en.wikipedia.org/wiki/IOS. No update information. Referred 02.06.2015

Wikipedia 2015b. Tizen. WWW document. http://en.wikipedia.org/wiki/Tizen. Updated 15.10.2015. Referred 17.10.2015

Windows Central 2016. Windows 10 mobile. WWW page. http://www.windowscentral.com/windows-10-mobile. Updated 22.03.2016. Referred 22.03.2016

Ziflaj, Aldo 2014. Native vs hybrid development. WWW article. http://www.sitepoint.com/native-vs-hybrid-app-development/. Updated 15.09.2014. Referred 09.06.2015