TÉCNICO LISBOA

WeatherIST

iOS application for detailed weather prevision
in Continental Portugal

Tiago João Alves Duarte

Dissertation submitted to obtain the Master Degree in

Information Systems and Computer Engineering

Supervisor: Prof. Arlindo Manuel Limede de Oliveira

Examination Committee

Chairperson: Prof. Mário Jorge Costa Gaspar da Silva
Supervisor: Prof. Arlindo Manuel Limede de Oliveira
Member of the Committee: Prof. Luís Miguel Veiga Vaz Caldas de Oliveira

June 2014

Agradecimentos

Para começar, gostaria de agradecer e dedicar o resultado desta dissertação à minha mãe, Odete, e ao meu pai, Ílidio, que sempre me apoiaram nos bons e maus momentos e que sempre me motivaram a apostar nos estudos e na educação. Gostava também de agradecer à minha irmã, Alexandra por ser, só de si, uma inspiração e motivação para terminar o mestrado.

Agradeço ao meu coordenador Prof. Arlindo Oliveira pela confiança depositada em mim, atribuindo-me um trabalho com bastante visibilidade para o instituto.

Agradeço também à Rosa Trancoso e ao Carlos Palma pela disponibilidade e o empenho que mostraram ao longo destes meses de trabalho no desenvolvimento e integração de serviços no METEO-IST.

Um especial agradecimento ao Gonçalo Cardoso, antigo colega de trabalho na PT Sistemas de Informação, pela ajuda que prestou no *design* e imagem da primeira versão da aplicação.

Agradeço à Nina, pelo seu apoio, paciência, dedicação e ajuda.

Agradeço a todos os amigos que fiz durante o meu percurso académico, em especial ao José Semedo, Nuno Antunes, Daniel Costa e David Possidónio.

Agradeço também a todos os meus colegas de curso que trabalharam comigo.

A todos, um muitíssimo obrigado.

Abstract

This work is the result of a need to move the meteorological forecast system developed by METEO-IST to an iOS application. METEO-IST is a weather computational server owned by IST that calculates with great accuracy the different weather conditions (rain, wind, humidity, etc.) anywhere within the Portuguese continental territory. The predictions are calculated frequently (at every 15 minute intervals) and exhibit a great precision, distinguishing it from other global meteorological systems that are currently available. Currently the system makes the forecasts available through the group's website. At the request of several users, the objective was to create a native iOS application for the iPhone that provides these forecasts by taking advantage of the device’s capabilities.

Resumo

Esta dissertação insere-se na necessidade de transportar o sistema de previsões meteorológicas actualmente existente, desenvolvido pelo grupo METEO-IST, para uma aplicação de iOS. O METEO-IST é um servidor de cálculo meteorológico, do IST, que calcula com grande precisão as diversas condições meteorológicas (chuva, vento, humidade, etc.) para qualquer local do território continental. As previsões são calculadas ao nível sub-horário (de 15 em 15 minutos) e exibem uma grande precisão, distinguindo-se das previsões meteorológicas globais que estão actualmente disponíveis. Actualmente o sistema disponibiliza as previsões através do website do grupo. A pedido de vários utilizadores, pretende-se assim criar uma aplicação nativa para o iPhone que disponibilize estas previsões tirando partido das capacidades do dispositivo.

“Here’s to the crazy ones. The rebels. The troublemakers.
The ones who see things differently.
While some may see them as the crazy ones, we see genius.
Because the people who are crazy enough to think they can change the world,
are the ones who do.”

Apple Inc.

Contents

List of Figures

List of Tables

Acronyms

PDA Personal Digital Assistant

PPI Pixels per Inch

HD High Definition

LCD Liquid Crystal display

OS Operating System

iOS iPhone Operating System

OS X Operation System X

IDE Integrated Development Environment

SDK Software Development Kit

1 Introduction

This project report presents the design and development of an iOS application called 'Weather IST'.

This app is primarily designed for people who usually get their weather forecasting information by accessing the METEO-IST website, and, its development is divided in two parts. The client side consists of the iOS device whose programming is done through Xcode using objective-C (a variant of C and C++) and iOS SDK. The other part is the web services that provide the weather forecasting data to the mobile device.

This report has 6 chapters. In Chapter 1, I will give a general overview of iOS devices, the reason why a mobile app was needed for METEO-IST and a survey of the most highly rated, and used, weather apps. In Chapter 2, I will describe the METEO-IST work and also the development made in order to support the app requirements. Chapter 3 will explore the technologies and tools used in the development of the app. Chapter 4 goes through the software cycles of the application development process, and also gives an overall insight on user satisfaction and social networking around the app. Chapter 6 concludes the report and describes what I have accomplished, further improvements to the app, and what I have learned.

1.1 Mobile Devices

Mobile communications are so integrated into our lives that many people already feel uncomfortable without a mobile phone. The time when mobile phones served only to make calls and send text messages is gone and the biggest reason was the arrival of the smartphones.
What are these smartphones and how do they contribute to a higher user satisfaction compared to previous devices?

A smartphone is a designation given to a mobile device with greater computing power (and connectivity). Close to the end of the twentieth century the first smartphones combined the functionalities of a PDA with those of a mobile phone.

Despite the growing evolution in hardware, the operating systems running on the phones were comparatively lagging behind in evolution and did not allow much customization by the end users even if they were already demanding more software choice and greater freedom of customization. Companies started to launch more applications with their devices such as games and utility applications (i.e. calculator, alarm, etc.). Nokia, for example, was made famous for putting the Snake game on their first devices. These early devices changed people's views of mobile phones and together with a decrease in prices and an increase in battery autonomy and network coverage led to a rise in sales.

A few years earlier, on a garage in Los Altos, California, two kids started a company called 'Apple Computer' that would change various industries and ultimately the world [7].

1.2 Apple

The company ‘Apple Computer’ was founded and established by Steve Jobs, Steve Wozniak and Ronald Wayne in 1976 to sell the first personal computer called Apple I. It was a complete microcomputer burned onto a circuit board with a built-in video terminal and a socket for 8 kilobytes of on-board RAM memory. By adding a TV set and a keyboard, a user had a fully interactive computer that was priced at about $660 USD.

Since then, ‘Apple Computer’ has revolutionized two industries: the computer industry, having developed the first personal computer and subsequent upgrades, and the music industry with the release, in April 2003 of iTunes, a software-based online digital media store that allows users to buy songs individually without any subscription fees.

In January 2007, Steve Jobs unveiled what would be their third revolution, the reinvention of the mobile phone with a device called the iPhone. His success was almost guaranteed. It revolutionized so many components (both in hardware, software and design) compared to previous phones that it sold 6.1 million units in the first 15 months after its release. In 2008, only one year after, ‘Apple’ was already one of the five largest mobile phone manufacturers by revenue [3].

1.3 iPhone

The expression “reinvention of the mobile phone” is not just a catch phrase. They did reinvent it. The first major change compared to older phones was the fact that the iPhone did not have a physical keyboard. Instead, it provided the keyboard on the screen whenever you need it (i.e. focus on a textbox) and its buttons were just the right size to use the best pointer we have, our finger. The second major difference was its 3.5 inch LCD touch screen, shown in Figure 1, which was able to render an image at 480 × 320 pixels with 163 PPI’s. It also had a 2 Megapixel’s camera, and a GPS, among other hardware features.

Despite all the hardware evolution, the software evolutions were also notable. It’s operating system called ‘iPhone OS’ at the time (and later changed to iOS), was also developed by Apple and therefore tightly coupled to the hardware which meant that it worked flawlessly with no breaks or errors which often occurred in other devices. Apple also decided to separate the phone from the carrier, which meant the company would decide which contents they wanted on the device, and therefore, the carrier would only be responsible for supplying the network to the device.

Figure 1: iPhone's 1st generation

The user interface was a touch-based control, which was one of a kind. It gave the user the freedom to use their fingers to interact with the device. It was developed in such a way that different touches or gestures on the screen were recognized and each gesture had a specific functionality, making it a multi gesture touch based device.

1.4 iPhone SDK

In March 2008, Apple introduced the iPhone SDK for developers to build applications for the iPhone as well as other Apple devices like the iPod Touch. It provided the developers with the same set of development tools and Application Programming Interfaces (APIs) that Apple uses for building native applications on the iPhone and iPod Touch. Within a week Apple had 100,000 downloads of the SDK [2]. The SDK is available through Apple's 'iOS Standard Developer Program' which is €80/year or the "iOS Enterprise Developer Program" (iDEP). The iOS Enterprise Developer Program is for apps which are not meant to be released and are only for institutional use. These do not require a membership fee but need to be approved by Apple.

The iPhone SDK comes as a single package that includes the iOS SDK, Mac SDK, and Xcode. Xcode is an Integrated Development Environment (IDE) that includes developer documentation, an Interface Builder (a tool to build graphical user interfaces) and an iPhone simulator. The package is

called the complete Xcode developer tool set for developing applications on the iMac, iPhone, and iPad and includes all the tools and frameworks required to build Mac OSX and iOS applications.

The packaging and distribution of apps can be done through the App Store.

1.5 App Store

The Apple App Store is a digital application distribution platform for iOS, developed and maintained by Apple. The service allows users to browse and download applications that were developed with Apple's iOS SDK. The apps can be downloaded directly to an iOS device or onto a personal computer via iTunes. Both the App Store and the iPhone SDK were released simultaneously as they are meant to complement each other. The App Store was made available to iOS first version through an update on iTunes and was made available natively on the second version of the operating system (released that same year) and all later versions.

Through the App Store, developers can publish their apps. The app can be priced by developer/seller preference or given away free. There are many applications available in the market ranging many different areas including games, business, education and weather. The idea that anyone can develop an app and sell it has led to an enormous number of apps in the market with a broad range of creative ideas.

1.6 Background

A METEO-IST user gets its weather information by visiting the website. Although it provides accurate information about weather forecasting it is not as fast and easy to use (usability wise) as it could be. For example, a user that enters the website and wants to get the weather predictions for his location has 3 options: choose from a list of districts, click directly on the map of Portugal or enter the location's coordinates (latitude and longitude) manually. Either way the user is then sent to a second screen that shows him the temperature for the next hours/days and if he prefers to see a different metric (i.e. rain or cloudiness) he has to click it, thus refreshing the page for the chosen metric. The information is scattered throughout the website and it requires many clicks before a user can get all the information he wants.

The application 'Weather IST' was designed to provide weather information to users in a fast (efficient) and easy way. It also takes advantage of iPhone features like local storage, for persistent application settings, or GPS to determine the user's location and give the respective weather information for it. In Chapter 4 I will discuss further the application design choices.

1.7 Similar Work

There are currently more than 400 weather applications on the App Store, half of them paid and the other half free. In order to understand what makes users choose some applications over the others, I decided to study the features and design of the top three paid and free weather applications.

1.7.1 Weather Live

Weather Live is the most highly rated paid application on the App Store. It has almost 21,000 reviews with an average rating of 4.5 out of 5. As you open the app for the first time you notice the thoughtful design and nice animations that help illustrate the weather conditions with the assistance of real images as background as shown in Figure 2. The application also takes advantage of the iOS notification system (used in native applications like the email app) to warn the user when the temperature drops below 0º Celsius or when there are forecasts of harsh weather conditions for the user's registered locations.

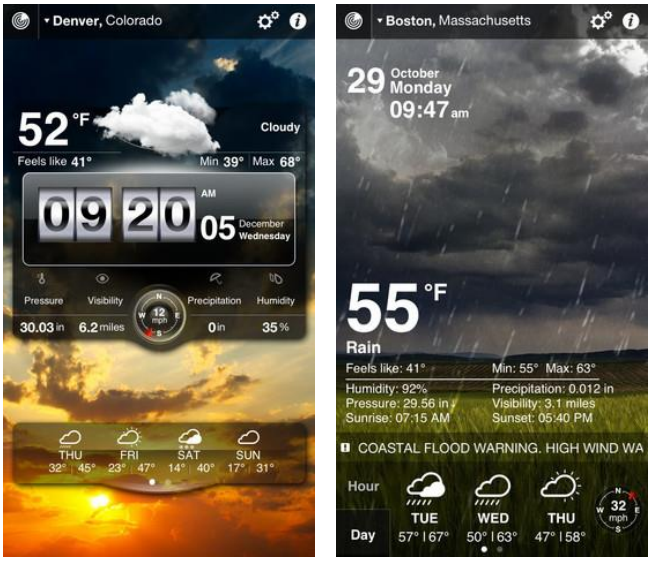

Figure 2: Weather Live application

The application allows users to see the weather forecast by hour (up to 24 hours) and day (up to 7 days). The available metrics include temperature, rain, humidity, wind velocity and direction, atmospheric pressure and visibility, and the app allows the user to change the units of some of these (metric versus imperial system).

1.7.2 Weather Genie

Weather Genie has almost 400 reviews and an average rating of 4 out of 5 stars. Just like Weather Live, this paid app tries to catch the user's attention by using HD images that illustrate the weather conditions with animations that represent for instance rain and snow as shown in Figure 3. The application shows hourly predictions, up to 24 hours, and daily previsions, up to 7 days. The available metrics are the same as in the previous application.

Figure 3: Weather Genie application

1.7.3 Weather HD

Weather HD is the third most highly rated paid application of the App Store with an average rating of 4 out of 5 stars. One of the key features of this app is that it contains animated radar maps, as shown in Figure 4, with the weather data being supplied by the National Oceanic and Atmospheric Administration (NOAA) which makes it very accurate. Another key feature is the ability to share the weather previsions on Facebook and see other people's shares. Similar to the previous applications, Weather HD uses HD images and animations and shows forecasts for 24 hours and 7 days for the most relevant metrics.

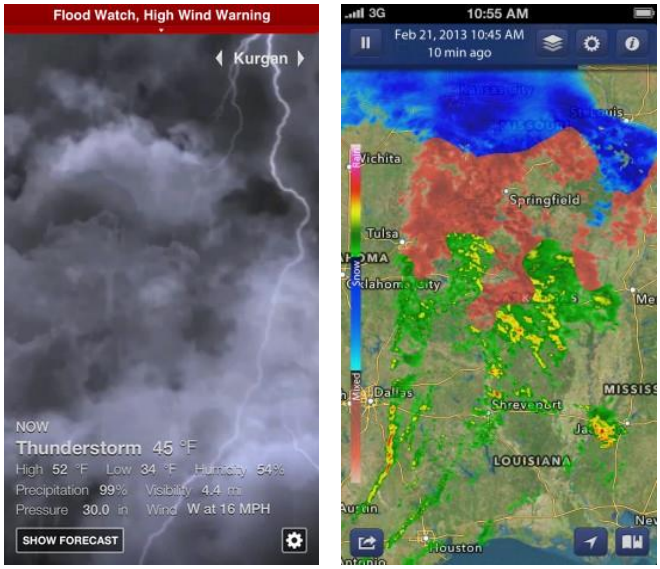

Figure 4: Weather HD application

1.7.4 Thermometer!

This free application is the most highly rated free application on the App Store. It has almost 300 reviews with an average rating of 4 stars out of 5. The application is based on a minimalist design, as shown in

Figure 5, and only shows predictions for the current day which include temperature, rain, atmospheric pressure and wind velocity and direction.

By being free to any user, and following the philosophy of Apple, the app contains publicity, which is the only way for developers to profit with free apps. Apple has developed the iAd program, which allows any developer to integrate ads into their application without having to develop their app around it.

Figure 5: Thermometer! Application

1.7.5 AccuWeather

AccuWeather has approximately 83,000 reviews with an average rating of 3.5 out of 5 stars. Unlike the other applications, it contains daily forecasts up to 15 days and the usual 24 hours. One of the key features of this application is that it can be integrated with the iPhone calendar, which allows a user to check the weather predictions without having to open the app. Another great feature the possibility to store favorite places on the device using the iCloud service, which allows the synchronization with other devices like an iPod touch or a Macintosh, and also allows the sharing of forecasts on Twitter and Facebook. In terms of the design, the app is well planned out although it revolves around a simplistic design more focused on the information it provides rather than its image, as shown in Figure 6.

Figure 6: AccuWeather application

1.7.6 The Weather Channel

The Weather Channel has about 360,000 reviews and an average rating of 3.5 stars out of 5. The application contains daily forecasts up to 10 days and 24 hours. Like AccuWeather the application design is very minimalist and focuses mainly on the carefully placed display of the metrics, as shown in Figure 7. However, it contains interesting features like dynamic backgrounds that change with the weather conditions and it allows users to choose their own background from their personal images. The application also contains a notification system that warns about harsh rain forecasts, hurricanes or even the amount of pollen in the air. It is also possible to share the forecasts on Facebook.

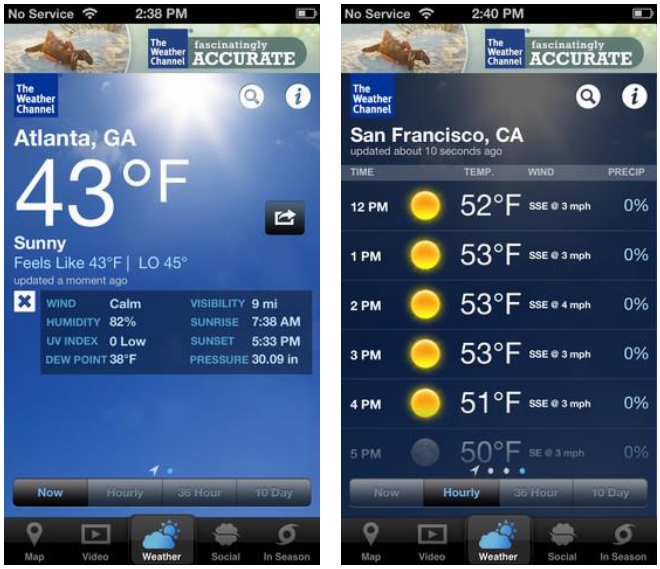

Figure 7: The Weather Channel application

1.7.7 Conclusions

Although existing applications already contain the most common metrics in weather forecasting (i.e. temperature, rain, etc.) they do not allow users to see them discriminated for the current and following days. This means users cannot have a perspective of the metric's evolution for the next few hours or days. One of the 'Weather IST' key features is that it will make the weather metrics independent which will allow the user to define and isolate the information that most matters to him. There are not optimal weather conditions for all users as each one has its own needs. For example, a firefighter on duty would search for a period of the day (or next days) with less wind velocity and more humidity, while a surfer would be interested in the sea wave's size and not really care about humidity or wind velocity.

Another advantage of our application is the precision and reliability of the forecasts that the METEO-IST servers produce for continental Portugal. The formulas behind the estimation of the forecasts are backed by years of scientific studies and are constantly being validated and improved. The next Chapter will focus on explaining the work done at METEO-IST as well as the developments made to support the app functionalities.

Although the app has many strengths, my goal is not to try and surpass the apps described above. My aim was to create an easy to use application with precise forecasts which the current METEO-IST website users can have as an alternative source of weather information. Chapter 4 will describe user satisfaction and compare it to my initial expectations.

2 METEO-IST

METEO-IST is a numerical weather prediction group founded at IST. It is part of the mechanical engineering department in the energy and environment section. Although this thesis does not involve forecasts calculations itself, but only the results, I describe in this section the evolution of numerical weather prediction in general and at IST.

2.1 Numerical Weather Prediction

Numerical weather prediction (or NWP) uses mathematical models of the atmosphere and oceans to predict the weather based on current weather conditions.

The first attempt at this system was made in the 1920's, but due to the lack of computer computational power the first realistic results were only produced in the 1950's [8].

There are a number of global and regional forecast models that are run in different countries and use current weather observations relayed from radiosondes[1] or weather satellites[2] as inputs to the models.

These mathematical models can be used to generate either short-term weather forecasts or longer-term climate predictions with the latter being widely applied towards understanding and projecting climate change.

[1] Units that measure various atmospheric parameters.

[2] Satellites that are primarily used to monitor the weather and climate of the Earth.

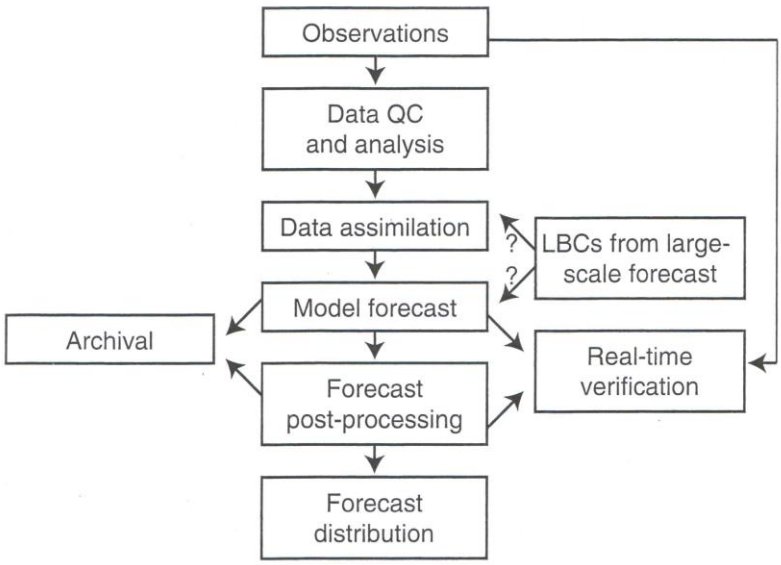

Figure 8: The various components of a NWP system.

The NWP used at IST is based on two models. The MM5 (short for Fifth-Generation Penn State/NCAR Mesoscale Model) and the Weather Research and Forecasting model (WRF) are offered free of charge and are supported by the United States National Center for Environmental Prediction (NCAR). Both models are suitable for a wide range of applications with a scale ranging from a few meters up to thousands of kilometers. Initial and boundary conditions to both models are provided by the Global Forecast System (GFS), a global spectral weather prediction model running at the National Centers for Environmental Prediction (NCEP) [9].

The MM5 software was initially developed in 1994, by the Pennsylvania State University (PSU) and was later on upgraded with various contributions from institutions and individuals until 2005.

The last version was released in December 2004 with the last known bug being fixed in October 2006. Today, the model is still supported although new developments are being incorporated into the new WRF model.

The NWP systems were first brought to IST by Prof. Delgado Domingos in 1995 using a cluster of 4 Intel PC's. He also developed the early scripts to run the required software for a viable and reliable operational model.

After exhaustive testing and configuration the model was first brought to public in 2001 where users could access the weather forecasts by visiting their website[3].

The website allowed users to pick any location on continental Portugal by providing a pair of coordinates (latitude and longitude) and it was the first system of its kind in Portugal to allow such a free use of a weather prevision system to get real time information of the most important metrics like temperature, rain, etc.

Initially, the model produced 72-hour forecasts with domains of 81, 27 and 9 kilometers, Figure 9, updated four times in the day (00Z, 06Z, 12Z and 18Z).

[3] http://meteo.ist.utl.pt

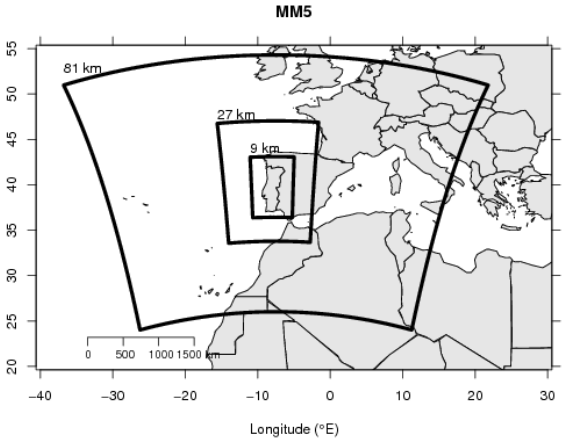

Figure 9: Weather forecast domains of the first MM5 model implemented in IST [9]

In 2007, with the upgrade to GFS and the introduction of a new NWP model, the WRF, both models began using the new GFS, as shown in Figure 10. The outer coarse domain of MM5 was dropped and the WRF model, now sufficiently mature to be operationalized, was added. WRF was implemented with nested grids of 9 km for Portugal, and a 3 km in a square domain with approximately 200x200 km in extent centered in the city of Lisbon. Today, each model produces 72-hour forecasts updated four times in the day (00Z, 06Z, 12Z and 18Z) and 180-hours forecasts updated each day at 00Z.

Table 1 shows the options used in each model and domain.

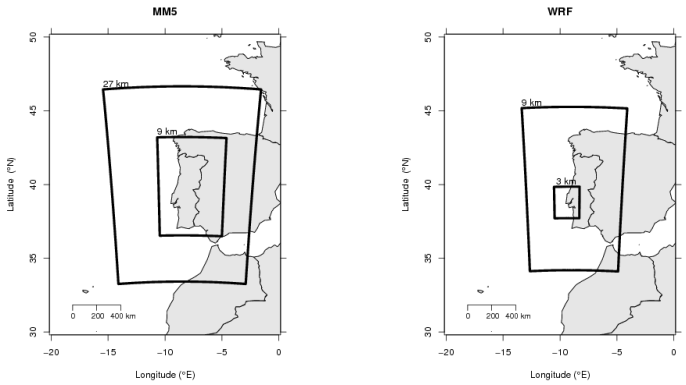

Figure 10: Domains in IST 3-day and 7-day forecast system for Portugal, since 2007 [9]

Model	GFS	Resolution (km)	NYxNX
MM5 3.7	1º	81	40x50
	-	27	55x40
	-	9	82x55
MM5 3.7	0.5º	27	55x40
	-	9	82x55
WRF 3.2	0.5º	9	135x80
	-	3	79x64

Table 1: Options used in each model and domain of the IST operational system [9]

3 Technical Details

This chapter provides a broad overview of the tools and technologies that are currently available for the development of applications in iOS.

Although this project does not utilize all the tools and techniques described in this chapter, the survey establishes the background for the technology used in this project.

3.1 iPhone Development Bundle

The iPhone Operating System (later called iOS) was originally developed for the iPhone and later extended to other devices produced by Apple like the iPod Touch, which is a music player, and the iPad, which is a tablet.

Figure 11: Devices using iOS

The first iPhone was released in June 2007 and since then it has evolved every year, both in software and hardware, with the release of new versions of its operating system and models. The iPhone is now in its seventh generation, with the iPhone 5S and the operating system is also in its seventh version, named iOS7.

Both the phone and the OS are developed by Apple and therefore the OS is not available for installation on devices other than those distributed by Apple. This tight connection between the hardware and software is something that Apple has always fought for and has distinguished this company from others like Microsoft or IBM [7].

Having an OS that works exclusively in a restricted number of devices makes it easier to assure performance and stability, something that Android, for example, is not able to do. It also makes the developer's work easier as he does not need to worry as much about the multiple devices in which the app would be run and the inevitable differences that would occur at the level of screen size or computational capability, for instance.

iOS is derived from OS X. The operating system used in Apple's desktop and laptop devices like the iMac and Mac Pro, respectively, and is similar to the UNIX operating system. The user's interaction with the iOS is made through gestures like tapping, pinching, reverse pinching and swiping. Since the release of the iPhone OS, various updated versions have been released containing new features and fixes for the previous version.

The home screen displays application icons and a dock at the bottom of the screen where users can pin their most frequently used apps (up to four). The home screen appears whenever the user unlocks the device or presses the "Home" button (a physical button on the bottom part of the device) while using another app, and it is the doorway to open any application in iOS. All the icons in the home screen are coherent in the way that they all represent a different application.

Although iOS is derived from OS X, one does not need to know how to program for OS X in order to develop iOS applications. I would classify iOS applications in three different categories.

System Applications

System applications are apps made by Apple that can talk to the underlying hardware directly. For example, switching the Wi-Fi on and off or installing a new app on your device is done by this type of applications. "Settings" and "App Store" applications are examples of these.

Native Applications

Native applications are apps also made by Apple but which are developed using the same layers of architecture (API's) as the ones the developers have access to. "Calculator" and "Notes" are examples of such applications.

Third Party Applications

Third party applications are apps done by any developer, other than Apple, and are limited in their use of some architecture layers. 'Weather IST' is a third party application.

3.1.1 iOS Architecture

iOS acts as an intermediary between the underlying hardware and the apps created for the iPhone. As stated above, apps do not talk to the underlying hardware directly. Instead, they communicate with it through an available set of system interfaces. These system interfaces are divided in four different layers, shown in Figure 12, that provide fundamental services and technologies in the lower levels and more sophisticated services in the higher levels.

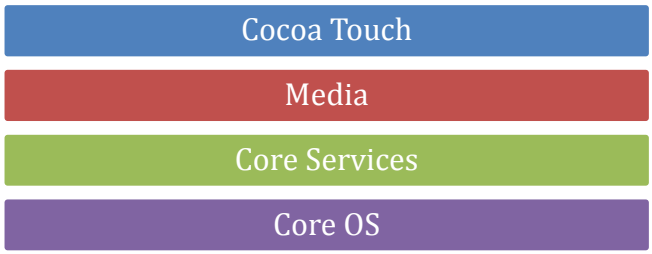

Figure 12: Layers of iOS [5]

Apple recommends the use higher-level frameworks over lower-level ones whenever possible. The higher level ones provide a more abstract interface that is less likely to change and have methods that can become deprecated. In these higher layers, abstractions also make the code smaller because they encapsulate potentially complex features that a developer does not have to take into account in most cases.

Core OS

The Core OS layer contains the low-level features upon which most of the other technologies are built. This layer is subject to change in every OS update and a developer should therefore only use it directly if he cannot access the desired technologies using the higher-level layers, for example, in situations where one needs to explicitly deal with security or to communicate with an external hardware accessory.

Core Services

The Core Services layer contains fundamental services for apps. It contains the Core Foundation and Foundation frameworks which define the basic types that all apps use like the 'View Controller' which is the class that almost every navigation view inherit from. It also contains the location services that allow an application to use the device's GPS as well as the iCloud, social media and networking.

Media

This layer contains the graphics, audio and video technologies developers use to implement multimedia in their apps.

This layer encapsulates the following technologies:

- Graphics Technologies – Defines high-level support for drawing images and Bézier paths and for animating the view's contents. It also provides advanced support for manipulating video and still images and 2D and 3D rendering.
- Audio Technologies – Provides the developer the ability to play and record high-quality audio in their applications.
- Video Technologies – Provides support for managing static video content and even streaming it from the internet. It also provides the tools needed for video recording for devices with appropriate recording hardware.

Cocoa Touch

The Cocoa Touch contains frameworks that define the appearance of the app and also provides the basic infrastructure and support for technologies such as multitasking, touch-based input (i.e. tapping or zooming), push notifications, etc. This layer is where the frameworks for the user interface reside.

3.1.2 IDE

Both the iOS SDK and Xcode come in a single package. Apple provides rich API documentation as a development guide and, for each API, they also provide code samples for reference. Most of the code in an iOS application is written in Objective-C.

3.1.3 Xcode

Xcode is an IDE that contains software development tools (all developed by Apple) that support the development of applications for OS X and iOS. It handles most of the project details and takes the project from inception to deployment containing documentation for every API it provides. The IDE contains an interface builder which is a tool that allows developers to create their own interface by dragging elements to the view and customizing them. It supports C, C++, Objective C, and Objective C++, among others. Xcode provides tools to manage the entire development workflow, from creating the application, testing, optimizing, and submitting it to the App Store. The figure below shows an example of 'Weather IST' being developed in Xcode.

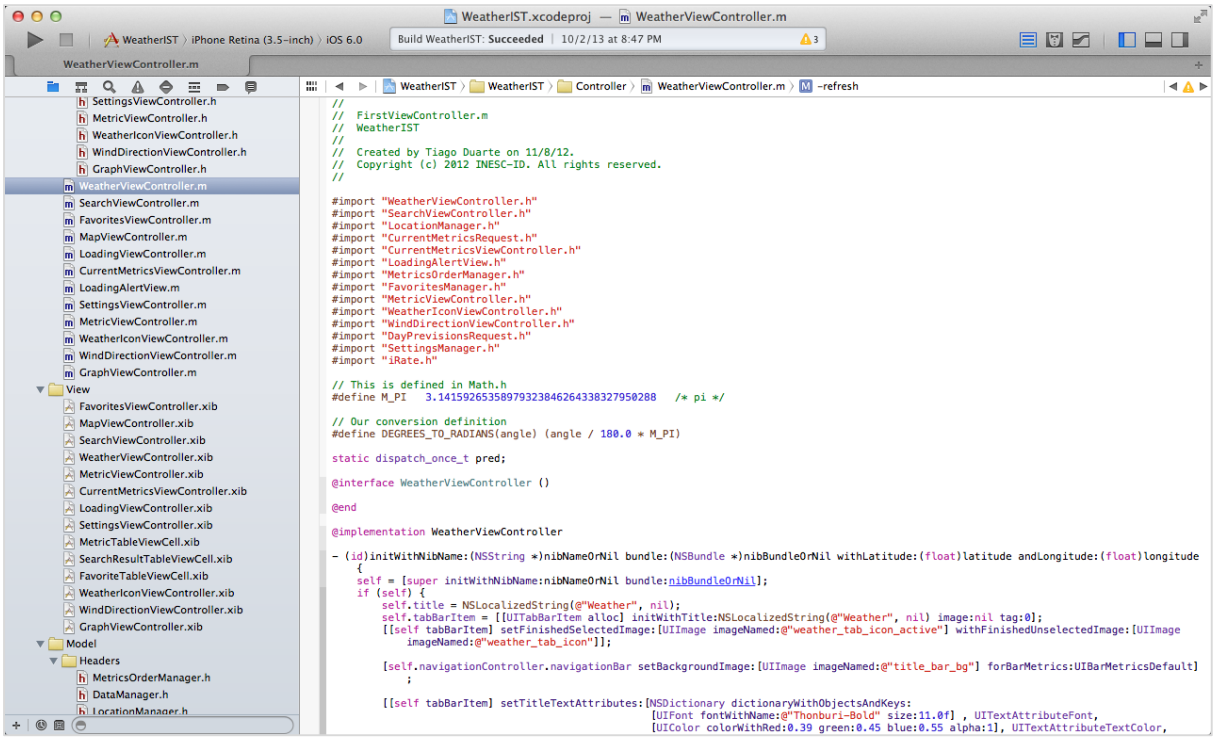

Figure 13: 'Weather IST' development in Xcode

Xcode allows us to:

1. Create, manage, build and deploy projects.
2. Specify the platform, the active target, the build configurations and the target requirements.
3. Navigate through project directories, libs and frameworks.
4. Debug a project locally using an iOS simulator or a physical device.

It also supports several source-code management systems like Sub versioning. For debugging, Xcode provides options like GCC (the GNU C compiler), LLVM-GCC compiler and the Clang compiler.

Xcode is now in its fifth version (released with the iOS 7 SDK).

3.1.4 Interface Builder

The Interface Builder provides collections of user interface objects to the developer. These user interface objects contain items like text fields, data tables, sliders and buttons and can be found in the "Object Library" of Xcode. These collections are completely extensible meaning any developer can create new objects and add them to Interface Builder.

It is also possible to configure the original objects in the user interface. A developer can do this by accessing the "Inspector" and modifying many attributes of the objects. It also contains other selectable panes from which he can set the configuration of the object at initial runtime.

An interface file is saved as a package that contains the interface objects and relationships between them (events, inherency, etc.). All of the objects used in the interface are archived (process also known as serialization) as an XML file or a property list file with a .nib extension. Upon successfully deploying and running an application, the proper NIB objects are opened and connected to the binary of their owning app [4].

3.1.5 Connections panel

In order to associate Objective-C code with an interface element a developer needs to somehow connect both elements. The connections panel is a context-specific window that shows up only when there are outlets or actions associated with a specific object. To associate a code declared property with an interface object, the developer must declare the property as an “IBOutlet” which will let him then connect that property to the object. Because the interface builder is integrated with Xcode, it knows about the outlets, actions, and bindable custom properties written in the custom classes. Hence, every time something changes in the custom classes, the interface builder detects it and updates itself.

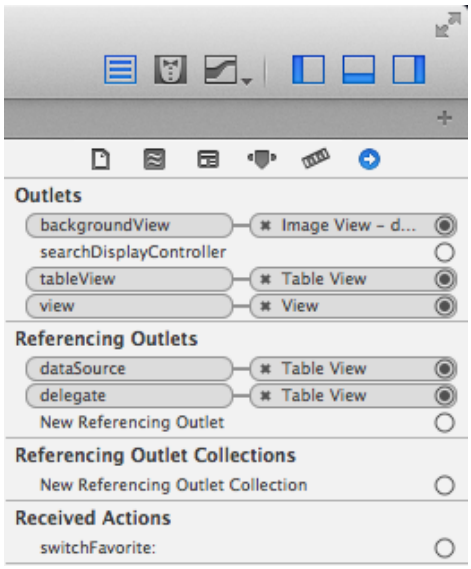

Figure 14: The Interface Builder connections panel

3.1.6 iOS Simulator Application

The SDK provides an iOS simulator that can emulate the iPhone on the Macintosh. When a developer wants to deploy an application he has to choose the target where he wants to deploy it to. If there is an iOS device plugged into the computer, and it is properly provisioned (the appendix contains a tutorial on how to achieve this), Xcode automatically adds it to the target options list and lets you deploy it directly onto the device with debugging ability. Although every developer should test their application in a physical device before submitting it to the App Store, it would be impractical to test it for all possible conditions. iOS Simulator allows a developer to simulate several iOS devices and versions of its

operating system. Each simulated software version is considered its own simulation environment, independent of others, with its own settings and files [6].

3.1.7 Performance tools

As the device has limited memory and computing power, it is important to test the application's performance so that it does not consume the entire power of the device thus leaving all other applications pendent. This really does not happen in iOS because the system allocates a maximum amount of memory that an app can occupy while it is running, and if that value is exceeded, the system shuts down the application and send the user to the home screen [1]. Nevertheless, it is important to remain below this threshold in order for an app to not crash and keep customers satisfied.

In Xcode 3.0, an application called 'Instruments' was introduced. It can run multiple performance testing tools simultaneously and view the results as a timeline-based graphical representation. It monitors CPU usage, memory leaks, garbage collection, disk reads and writes, thread activity, etc. The application is depicted in the figure below.

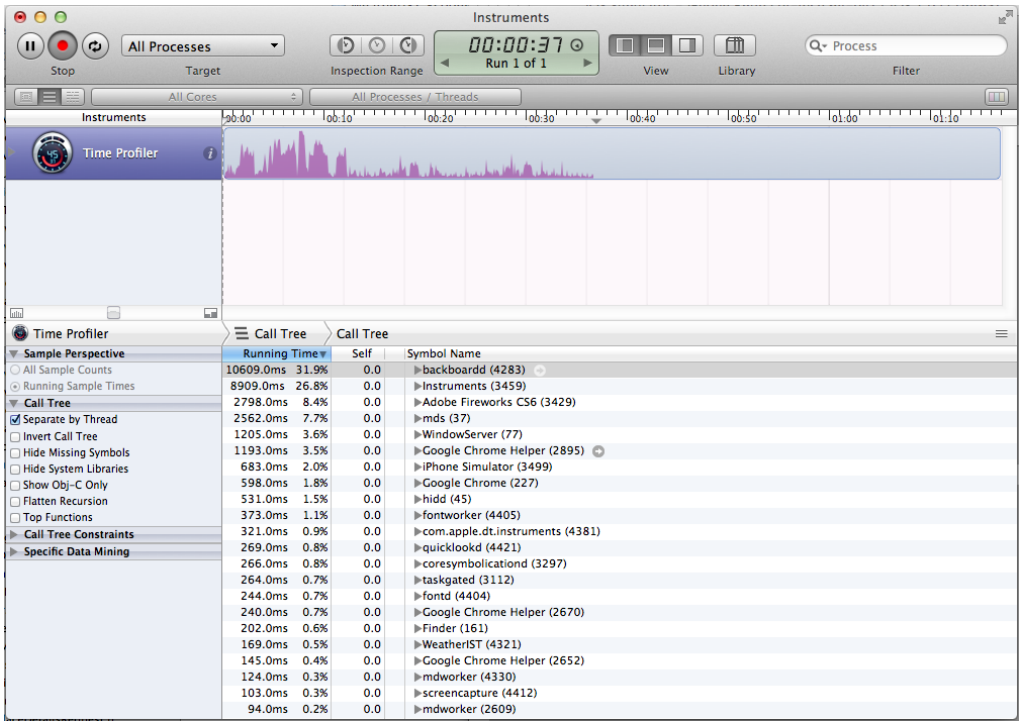

Figure 15: The instruments Application

There are also third party tools that can be used to debug the application.

4 ‘Weather IST’ Application

This chapter provides details on the various phases of the software development lifecycle. Given that the application had two major versions, I decided describe the details of each version in separate sections.

4.1 Version 1.0

4.1.1 Requirement Analysis

The idea of a mobile application for METEO-IST had been first suggested, by users of the website that wanted a portable version of the forecast service. The website’s interface is not very user friendly when used on a mobile device mainly, because it requires a lot of zooming and dragging just to set the right variables (i.e. location of the forecasts). Another important issue is that in a mobile device literally every byte counts. This is due to two reasons: firstly, almost every mobile data plan is limited to a maximum usage, and secondly, the network speed is slow, compared to what we can find in a LAN or Wi-Fi connection. This means that when a user accesses the website to get his forecasts he downloads more than just the information he wants to, because the response comes with the whole HTML of the webpage which includes elements that do not add any value (i.e. site navigation, footer, etc.) and only make the response bigger in size meaning that it also takes more time to load.

To start a requirement analysis, the first thing I did was to access the site and navigate through it as a regular user.

Like in any weather forecast system the workflow starts with the user inputting a location. In METEO-IST’s website there are three ways for a user to introduce his location: by selecting a district from a list (which includes all districts in Continental Portugal), entering a pair of location coordinates (latitude and longitude) in the correspondent textboxes, or clicking anywhere in the provided map of the country which then fills the coordinates automatically, as shown in Figure 16. One could think that having a coordinate system for a location choice is a bit exaggerated but, as mentioned in Chapter 2, METEO-IST servers use a 9 km grid and most districts are wider than that, so by allowing a user to choose a specific location, METEO-IST is also taking full advantage of its forecast data.

Figure 16: Location picker in METEO-IST's website

After a user picks his location, and clicks 'Submit' the page reloads with the forecasts for the chosen location, showing the temperature (by default, the first metric displayed). One thing I noticed (that was different from any other weather websites I had visited) was that the forecasts were all divided in different and independent metrics (i.e. temperature, cloudiness, etc.) which I thought was really good because it allowed a user to choose what is more relevant to him instead of providing the standard set of results. One down side of this division is that a user has to click and refresh the page as many times being as the number of metrics he wishes to see, also meaning that he cannot have a fast overview of all the metrics for a specific time.

Each metric forecast is shown in graph, which gives the user a nice overview of the evolution but can become quite hard to read for users that are not so comfortable with Cartesian coordinate systems.

To summarize, after the survey, some of the initial features to be included were:

- Search places in continental Portugal: I wanted a service that would allow a detailed location picking (to take advantage of the 9 km grid) but without having to input any coordinates or pick a location on the map. The service would allow to type a street name, city or district and give suggestions of places that matched what was being typed (i.e. an autocomplete function).
- User's current location: Using a mobile device like iPhone, with GPS capacity, allows us to know the user's location and give him the respective forecasts.
- Favorite places: We can take advantage of the local storage system of the device and store the user's favorite locations so he can quickly access them whenever he wants.

- Graph view: I thought the graphs were really helpful so I also wanted to keep this feature and allow a user to switch between this and the list view.
- Weather icons: Using the same forecast data as the website I wanted to improve on some of the basic features. One thing that users most like, when checking the weather, are icons that represent the current (or future) conditions because it allows a quick perception of the forecasts.
- Weather information: I started by visualizing an initial screen that would give all the metrics for the period (hour and day) they were being searched for, and then let the user navigate through each one to get more information on that metric (i.e. check the metric for the next few hours or days).
- Map view: Taking advantage of the Google Maps API (native on the iOS SDK) I wanted to let the user see their favorites places in a map, and, from there, access the forecasts directly for any of those places.
- Settings view: The user would be able to change the temperature and wind speed units, and I would use the local storage system to save these preferences.

4.1.2 Use Case Analysis

After the desired features of the software were defined, I started writing the Use Cases for each functionality. User here means any user that uses the app. In this context, view means the screen of the device like the page of a website. The Use Case for each feature is described below and its respective mockup can be found in the Appendix.

In each Use Case the precondition & alternative scenario will be the same hence will not be written about unless different.

4.1.2.1 Welcome/Weather View

Description: The user opens the application for the first time and gets the welcome/weather view.

Precondition: The user has downloaded the app successfully onto the device and the application is launched successfully.

1	The user successfully launches the application.
2	The app presents a warning asking his permission to use his location services (GPS).
3	The user taps on **Yes**.
4	The app returns a view showing the forecasts for the user's current location.

Table 2: Welcome/Weather View – Main Success Scenario

3	The user taps on **No**.
4	The app returns a view for location search.

Table 3: Welcome/Weather View – Alternative Scenario

4.1.2.2 Search View

Description: Searching for ‘Lisbon’ and checking the forecasts for it.

1	The user taps the search option.
2	The app returns a view with a text field to type a location.
3	The user types ‘Lisbon’ in the text field.
4	The app returns a list with two matched results.
5	The user taps the first one.
6	The app returns a view with the forecasts for the chosen location.

Table 4: Search View – Main Success Scenario

3	The user mistypes ‘Lisbon’ in the text field.
4	The app returns a list with no results.
5	The user corrects the word.
6	The app returns a list with two results.
7	The user taps the first one.
8	The app returns a view with the forecasts for the chosen location.

Table 5: Search View – Alternative Scenario

4.1.2.3 Map View

Description: The user wants to check the forecasts for a favorite place by going to the map view.

Precondition: The user has added, at least, one location as a favorite.

1	The user taps the map option.
2	The app returns a view with a map containing a pin for each place the user has set as favorite.
3	The user taps on one pin.
4	The map presents a tooltip above the clicked pin, with the favorite name and location.
5	The user taps the tooltip again.
6	The app returns a view with the forecasts for the chosen location.

Table 6: Map View – Main Success Scenario

4.1.2.4 Favorites View

Description: The user wants to check the forecasts for a favorite place by going to the favorites view.

Precondition: The user has added, at least, one location as a favorite.

1	The user taps the favorite's option.
2	The app returns a view with a list containing all the favorite places the user has added.
3	The user taps on one item.
4	The app returns a view with the forecasts for the chosen favorite.

Table 7: Favorites View – Main Success Scenario

4.1.2.5 Remove a favorite place

Description: The user wants to remove one of his favorite places.

Precondition: The user has added, at least, one location as a favorite.

1	The user taps the favorite's option.
2	The app returns a view with a list containing all the favorite places the user has added.
3	The user taps on the 'Edit' button.
4	The app reloads the list with a delete button on each item.
5	The user taps on the delete button in one item.
6	The app removes the item from the list.
7	The user taps on the 'Done' button.
8	The app changes the list to the normal state.

Table 8: Remove a favorite place - Main Success Scenario

4.1.2.6 Settings View

Description: The user wants to change the wind velocity unit from m/s to Km/h.

1	The user taps the settings option.
2	The app returns a view with two segmented controls that allow changing the temperature and wind velocity units.
3	The user taps on Km/h option in the wind velocity unit control.
4	The user goes to the favorites view and clicks on one of the favorites.

5	The app returns a view with the forecasts for the chosen favorite with the wind velocity displayed in Km/h.

Table 9: Settings View – Main Success Scenario

4.1.2.7 Forecasts for the following hours

Description: The user wants to check the cloudiness forecasts for the following hours on a favorite location he has named 'Home'.

1	The user taps the favorite's option.
2	The app returns a view with a list containing all the favorite places the user has added.
3	The user taps on a favorite called 'Home'.
4	The app returns a view with the forecasts for the chosen favorite.
5	The user taps on the cloudiness item.
6	The app returns a view with the cloudiness forecast for the next few hours.

Table 10: Forecasts for the following hours – Main Success Scenario

4.1.2.8 Forecasts for the following days

Description: The user wants to check the temperature forecast for the following days on a favorite location he has named 'Home'.

1	The user taps the favorite's option.
2	The app returns a view with a list containing all the favorite places the user has added.
3	The user taps on a favorite called, Home.
4	The app returns a view with the forecasts for the chosen favorite.
5	The user taps on the temperature item.
6	The app returns a view with the temperature forecast for the next few hours.
7	The user taps on the 'Day' button.
8	The app refreshes the view and shows the temperature (max and min) forecasts for the following 7 days.

Table 11: Forecasts for the following days – Main Success Scenario

4.1.2.9 Forecast graph

Description: The user wants to check the graph of rain forecast for the next day on a favorite location he has named 'Home'.

1	The user taps the favorite's option.
2	The app returns a view with a list containing all the favorite places the user has added.
3	The user taps on a favorite called 'Home'.
4	The app returns a view with the forecasts for the chosen favorite.
5	The user taps on the rain item.
6	The app returns a view with the rain forecast for the next few hours.
7	The user taps on the 'Day' button.
8	The app refreshes the view and shows the rain forecast for the following 7 days.
9	The user taps on the second item (which is the following day).
10	The app returns a view with a graph showing the rain forecast evolution from 00:00 to 23:00.

Table 12: Forecast graph – Main Success Scenario

4.1.2.10 Reorder metrics in the weather view

Description: The user wants to arrange the metric list so that the rain metric is in first position, temperature is in second and wind velocity is in third.

1	The user taps the weather option.
2	The app returns a view containing the forecasts for the last place he viewed.
3	The user taps the 'Order' button.
4	The app reloads the list with an order button on each item.
5	The user taps and drags the rain item to the first position on the list.
6	The user taps and drags the temperature item to the second position on the list.
7	The user taps and drags the wind velocity item to the third position on the list.
8	The user taps the 'Done' button.
9	The app changes the list to the normal state.

Table 13: Reorder metrics in the weather view – Main Success Scenario

4.1.3 Design

The application is designed in such a way that it is efficient and does not use a lot of processing or computing power of the mobile device, as all the complex operations are made on the METEO-IST's servers.

4.1.4 Application Architecture

The iOS application will behave as a client in the proposed architecture and will use METEO-IST's servers as the weather forecast provider. It will be necessary to define a new service interface that will support the application requirements, and also that conforms to the mobile paradigm (i.e. fast and small responses).

The Client-Server model is a computational model that separates one from the other. Each instance of a client can send data requests to any of the available servers (this load balancing issue is transparent to the client) and waits for a response.

The new services will be available on the Internet, although the weather forecast data file, which is generated by the METEO-IST's servers, will only be available on the IST intranet meaning that the client application ('Weather IST') will only be accessing the interface that is available on a public machine inside the IST network. Because this machine is inside the network, it will then have connectivity and permission to read the data file and return the results. Figure 17 illustrates the architecture.

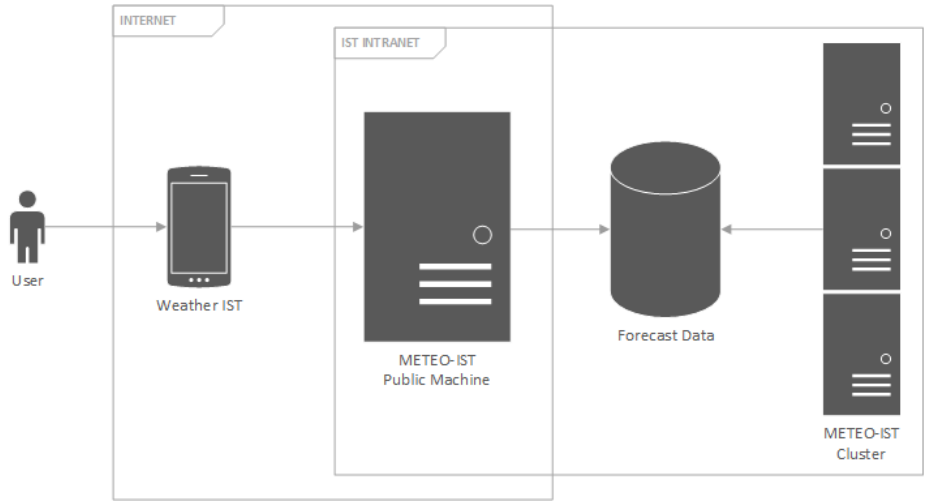

Figure 17: Top level architecture for 'Weather IST'

The following section will describe the services that were developed in METEO-IST in order to support the application requirements.

4.1.5 METEO-IST's Web Services

The services for this version were designed by me (request and response) and developed by Rosa Trancoso of the METEO-IST group. The services were developed in PHP which is a server-side scripting language designed for web development but is also used as a general-purpose programming language. The service's responses are all written in JSON, which is derived from the JavaScript scripting language and is used for representing simple data structures and associative arrays.

Even though each service serves a different purpose, the way the application builds the arguments and sends them is the same for all. All services have three arguments in common: a date (which can include a time or not), a latitude and a longitude. Before the requests are sent, the app gets the device current date and time (using native iOS API) and converts it to UTC[4]. This conversion ensures synchronization and precision in the forecasts. The app also gets the user's coordinates either by doing a search (which I will detail below), or by getting his location using the GPS (if he allows it). The coordinates are then rounded to floating point's numbers with a precision of 4 decimal places and put in the request. This process happens for all requests made throughout the app because these attributes are mandatory since as the forecasts are time and location specific.

Another attribute required in some services is the metric, and, in order to identify it, between the two endpoints, we had to define an enumerator that would be the same in the app and in the web services. Below, in Table 14, is the enumeration we defined.

Even though this protocol is required it does not limit the extensibility of the app. The code is made in such a way that if a new metric was implemented in the response, the application would not crash but simply ignore it as it would not recognize it. By just updating the enumerator on the app it could start parsing and displaying it.

Metric name	Description
temp	**Temperature** Celsius (ºC)
hum	**Humidity** Percentage (%)
rain	**Rain** Millimeters (mm)
cloud	**Cloudiness** Percentage (%)
wind_vel	**Wind Velocity** Meters per second (m/s)
wind_dir	**Wind Direction** Degrees from north (º)
atm_press	**Atmospheric Pressure** Millibars (mbar)

Table 14: Metrics enumerator – Used for identification with the web services

[4] Coordinated Universal Time

4.1.5.1 QuickPrevByDate.php

This was the first service to be designed (and developed) as it supplies the information I need for the initial app screen. I called it quick because that is what I wanted: a quick service so the app would not take too long to give an output. This service receives a location, sent in two single parameters (latitude and longitude), and a date and time, and returns the forecast, for each metric, for the period set. Because we want the metrics for the current hour (of the device), the date is sent in the format, 'yyyyMMdd_HH', where 'yyyy' is the year, 'MM' the month, 'dd' the day and 'HH' the hour. All of them have a leading zero (if necessary) so that the service always parses it correctly. Note that we do not send the minutes, because they become irrelevant since the forecasts are calculated by the hour and then interpolated. For example, for a device with the following date: October 1, 2013 at 1:37 the date parameter would be sent as '20131001_01'. Table 15 describes the service interface and response format.

The response is a key valued pair named 'response'**,** which is the root object. Inside it, we have a key for each metric the app knows, and for each key we have another object that gives us information about that metric. Note that the keys are semantically similar to the information they hold so they can be easily read. The temperature is the only object that has more than the 'now' key, and that is because there are two important values associated with it, aside the current value, which are the maximum and minimum temperature for the requested day. Although this information is not hour specific, it allows us to fill the welcome screen with all the information we need without having to make another service call just to display those values. This makes the app faster and less network consuming.

Parameter name	
Lon	Longitude (float with four decimal places)
Lat	Latitude (float with four decimal places)
Date	Date (string with the format yyyyMMdd_HH)
Example Call	
HTTP	PrevByDateAndMetric.php?lon=8.9341&lat=38.1294&metric=temp&date=20120813
Response example	
JSON	{ response : { temp : { max : 24, min : 12, now : 21 }, hum : { now : 30 }, rain : { now : 0 }, ..., wind_vel: { now : 5.4 } } }

Table 15: QuickPrevByDate.php – Service Interface

4.1.5.2 PrevByDateAndMetric.php

This service returns forecast predictions by date and metric. Looking back at the Use Cases, in section 4.1.2.7, the user taps on a metric and the app returns a view with the forecasts for the next few hours for that metric. This service was designed to support that feature. The service receives a latitude, a longitude, a date in the format 'yyyyMMdd' (the hour is not required for this request), and the metric name. Below, a table describes the interface.

Parameter name	
Lon	Longitude (float with four decimal places)
Lat	Latitude (float with four decimal places)
Date	Date (string with the format yyyyMMdd)
Metric	Metric name (string from the enumerator)
Example Call	
HTTP	PrevByDateAndMetric.php?lon=8.9341&lat=38.1294&metric=temp&date=20120813
Response example	
JSON	{ response : { points : [15, 14, ... , 16] } }

Table 16: PrevByDateAndMetric.php – Service Interface

The response has a root object named ‘response’ and its value is an array of exactly 24 values that correspond to the hours from 00:00 to 23:00. The application parses the array and shows all the values that have an index (in the array) equal or greater than the current device hour, in UTC. This will display the metric value for the current and next few hours until the end of the day.

This service also supports the graph feature (section 4.1.2.9) because it gives us all the necessary points to draw a chart, being that the X axis is the hour (from 0 to 23) and the Y axis is the metric value.

4.1.5.3 MedianPrevByDays.php

This service returns forecast predictions for the following days for all the metrics. It receives a latitude, a longitude, a date in the format ‘yyyyMMdd’, and the number of days (of forecasts) we want the service to return being that 7 is the maximum as the data file does not contain forecasts beyond that. Below is a description of the service interface.

Parameter name	
Lon	Longitude (float with four decimal places)
Lat	Latitude (float with four decimal places)
Date	Date (string with the format yyyyMMdd)
Days	Number of days to return (integer greater or equal then 0 and less or equal then 7)
Example Call	
HTTP	MedianPrevByDays.php?lon=8.9341&lat=38.1294&date=20120813&days=3
Response example	

JSON	{ response : [{ temp : { max : 24, min : 12, median : 21 }, hum : { median : 30}, … , rain : { sum : 0 } }], …, [{ temp : { max : 22, min : 12, median : 18 }, hum : { median : 30}, … , rain : { sum : 0 } }] }

Table 17: MedianPrevByDays.php – Service Interface

The response has a root object named ‘response’ and its value is an array with a length equal to the requested number of days plus one. For example, if the request was made with the date set for October 1, 2013 and the parameter ‘days’ set for a value of 3, the response would be an array with the metrics median values for October 1, 2, 3 and 4. Note that for the temperature, the service also returns the maximum and minimum value for each day (because it is a very common information) and for rain it returns the sum of the hour values instead of the median, because for rain we thought the median would not be very useful to the user. This service supports the Use Case described in section 4.1.2.8.

4.1.6 Other Web Services

With the METEO-IST’s services described above, I was able to support all the app features related to forecasts but I was still missing a way of doing location searches. I ended up asking a question to myself: what is the place where you search a street and it rarely misses? Google Maps. In fact, you can search streets, cities and districts, and they have an open API called Google Places, which is free with a usage limit of 100.000 requests every 24 hours.

4.1.6.1 Google Places – Autocomplete

Reading the Google Places API I found a service called ‘autocomplete’ that given a string of text, returns (at most) five results that match the search. Below, a table describes the interface of the service.

Parameter name	
Input	The text string on which to search
Types	The types of Place results to return
Language	The language in which to return results
Key	Indicates whether or not the Place request came from a device using a location sensor (true or false)
Components	A grouping of places to which you would like to restrict your results
Example Call	
HTTP	https://maps.googleapis.com/maps/api/place/autocomplete/json?input=Lisboa&types=geocode&language=pt&key=appKey&components=country:pt
Response example	

JSON	{
	"status": "OK",
	"predictions": [{

	"description": "Lisboa, Portugal",
	"id" : "691b237b0322f28988f3ce03e321ff72a12167fd",
	"reference": "CiQYAAAA0Q_JA...kT3ufVLDDvTQsOwZ_tc",
	"terms": [{
	"value": "Lisboa",
	"offset": 0
	}, {
	"value": "Portugal",
	"offset": 7
	}],
	"types": ["geocode"],
	"matched_substrings": [{
	"offset": 0,
	"length": 5
	}]
	}, …]
	}

Table 18: Google Places API Autocomplete – Service Interface

The idea is to call the service every time the user writes something in the search text field so it gives the idea of autocomplete and refreshes the results for the new string. In the 'input' parameter we send what the user wrote. The 'type' parameter specifies the result types we want. This could be for instance 'locality' or 'sublocality' but for our purposes I chose a type called 'geocode' which returns all geocoding results (results with coordinates). This is more interesting because it allows the search of real places instead of only streets (i.e. a person could search for 'Oceanário Vasco da Gama' and have weather forecasts for that location without even knowing where it is). This feature comes in handy if we think about tourists.

The 'language' parameter defines the language in which the results are returned. Given that our application is available in English and Portuguese, the value will be either 'en' or 'pt' depending on the user's iOS language. The 'key' parameter is the application identifier. It is used internally by Google to manage usage limits and statistics. Finally, we have the 'components' parameter that allows us to define the area for the results. Since METEO-IST's servers only generate forecasts for Continental Portugal this parameter really comes in handy because it restricts the results to this part of Portugal. Note that Google does not differentiate continental Portugal from insular territory so the application still has to treat error responses (from METEO-IST's services) and show a message accordingly saying it cannot generate forecasts for the selected location.

The implementation of this feature is depicted using a sequence diagram in Figure 18.

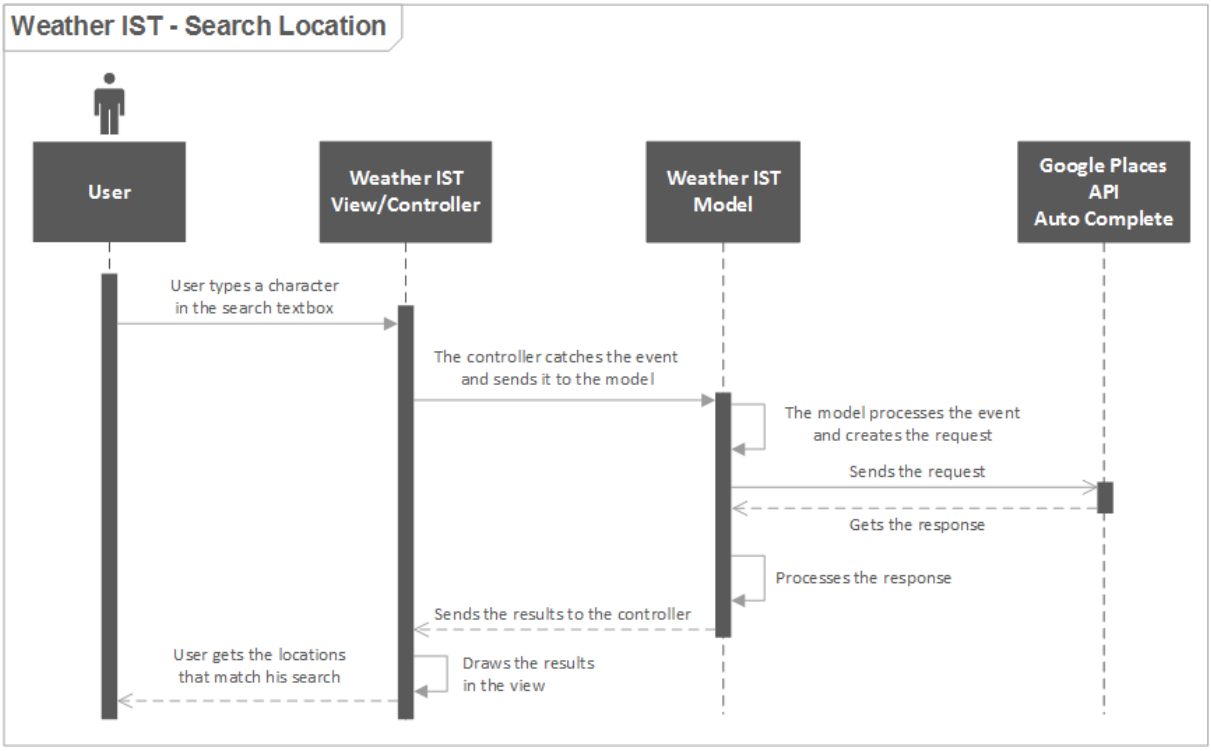

Figure 18: 'Weather IST' search location – Sequence Diagram

4.1.6.2 Google Places – Place Details

In the previous service note that the response does not contain a latitude or longitude, so if the user clicks that result what happens? Google does this because it wants to have a fast autocomplete service that returns the minimum amount of information. It really makes sense since the service can be called 6 times just by typing the word 'Lisbon'. When a user taps a result, we have to call another service from Google Places called 'place details'. It receives the place's id and returns more information including the latitude and longitude. The table below describes the service interface.

Parameter name	
Reference	A textual identifier that uniquely identifies a place
Key	Indicates whether or not the Place request came from a device using a location sensor (true or false)
Example Call	
HTTP	https://maps.googleapis.com/maps/api/place/details/json?reference=691b237b0322f28988f3ce03e321ff72a12167fd&key=appKey
Response example	

JSON	{
	"html_attributions" : [],
	"result" : {
	…. ,

	"geometry" : {
	"location" : {
	"lat" : -33.8669710,
	"lng" : 151.1958750
	}
	}
	….
	}
	"status" : "OK"
	}

Table 19: Google Places API Place Detail – Service Interface

Figure 19 shows the implementation of this feature depicted using a sequence diagram.

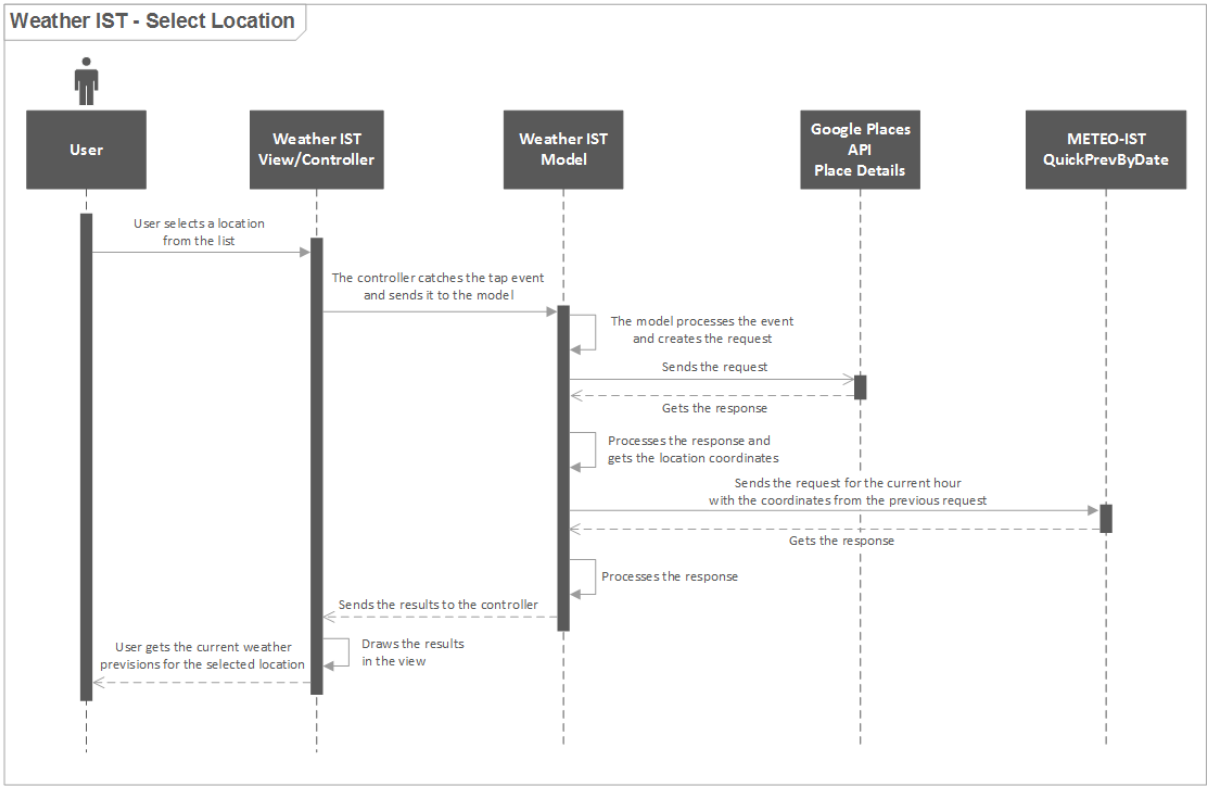

Figure 19: 'Weather IST' select location – Sequence Diagram

4.1.7 Development

To begin an Xcode project, we must start by selecting a template from eight possible choices. The differences between these templates are simply the initial classes that Xcode automatically generates with the chosen template, and the default packages that already come imported. Figure 20 shows the Xcode template window.

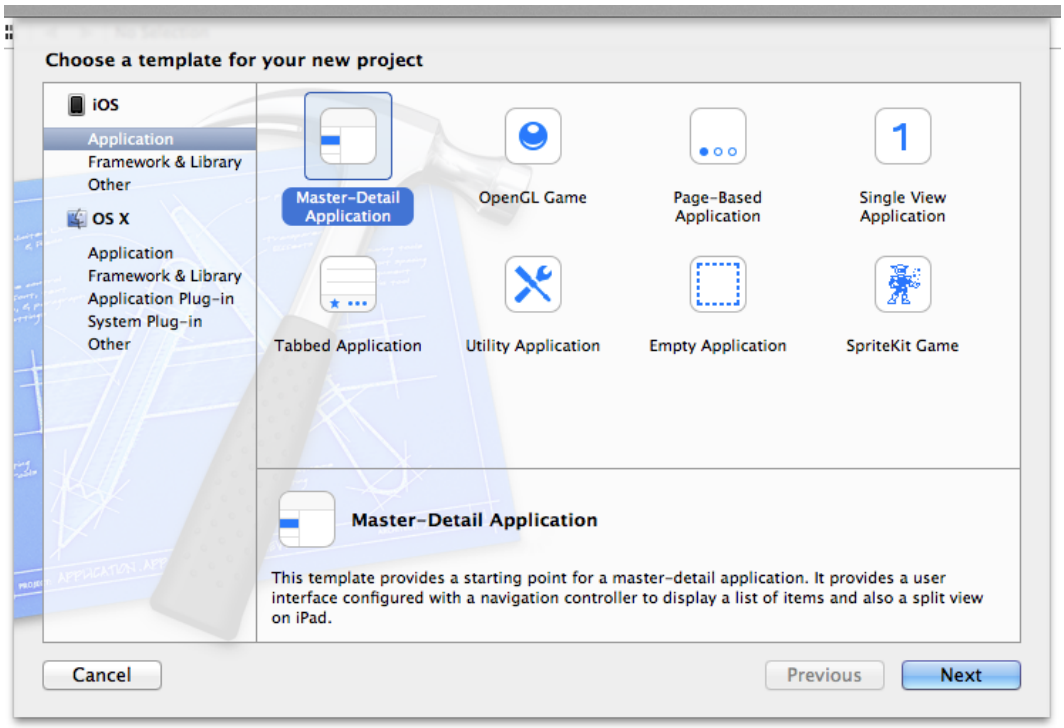

Figure 20: XCode template selection when creating a new project

Master-Detail Application Template

This template provides a starting point for a master-detail application. It provides a user interface configured with a navigation controller to display a list of items and also a split view on iPad.

OpenGL Game Template

This template provides a starting point for an OpenGL ES-based game. It provides a view into which you render your OpenGL ES scene and a timer to allow you to animate the view.

Page-Based Application Template

This template provides a starting point for a page-based application that uses a page view controller.

Single View Application Template

This template provides a starting point for an application that uses a single view. It provides a view controller to manage the view and a storyboard or nib file that contains the view.

Tabbed Application Template

This template provides a starting point for an application that uses a tab bar. It provides a user's interface configured with a tab bar controller and view controllers for the tab bar items.

Utility Application Template

This template provides a starting point for a utility application that has a main view and an alternate view. For the iPhone, it sets up an ‘Info’ button to flip the main view to the alternate view. For iPad, it sets up an ‘Info’ bar button that shows the alternate view in a popover.

Empty Application Template

This template provides a starting point for any application. It provides just an application delegate and a window.

SpriteKit Game Template

This template provides a starting point for a SpriteKit Game.

When I prototyped the interface I had not begun to study the iOS SDK and XCode so I really did not have the technical information necessary to corroborate my choices. I was simply an iPhone user and from that experience, I decided to design the application as a tabbed application.

The iPhone app development scenario is in many ways different from other scenarios like Android or Windows development. By learning iOS development I began to understand why the apps could look so different but have that same feeling of similarity between them, and these templates are the reason for that. Their goal is to define your app base navigation so that a new user knows how to handle and navigate through your app even if he is a first time user.

Even the Apple apps are designed using this formula. One app that uses the same template as ‘Weather IST’ is the App Store. A tabbed application is really handy when you have more than two views (screens) and you want the user to easily access them. The tab application allows a user to switch from one view to another without losing the application state. That is exactly what I wanted. For example, if a user is checking the weather forecasts, on Tab 1, and wants to go to the app settings, on Tab 5, he can simply tap the fifth tab, change the settings and go back to the weather forecast by clicking the first tab. Now the weather forecasts reflect the changes he has made, whether it was a change from Celsius to Fahrenheit, on the temperature, or a change from m/s to Km/s in the wind velocity unit.

Despite the template, they all have in common the ‘UIApplicationDelegate’. The ‘UIApplicationDelegate’ is a protocol that declares methods that are informative about the state and events occurring in the application like when the application terminates, when the memory is low or when the application has finished launching. By implementing these methods we can respond and interact with the system events as they occur. This class is always the starting point of any application.

The navigation between views is very easy to implement for the developer. He has been given a set of methods in the SDK that will do all these animations and transitions with a simple call to a push view, to make a view appear or pop view to make a view disappear. It is also possible to programmatically switch between tab views without the user having to touch the tab controls. In our application, we use this feature every time the user clicks on a search result or a favorite. We send him

from tab 2 (search), tab 3 (favorites) and tab 4 (map) to tab 1 (weather forecasts) automatically updating the tab controller state to reflect the changes.

Custom view controllers are created according to the requirements of the app to support each feature. Nib files are also created to design the user interface with views, user controls, attach target to the controls in the view controller, etc.

The user interface can be created using the Interface Builder or drawn programmatically. In the application the nib file is created for views when necessary to keep functionality and design separate. The main custom view controllers (.h & .m files) implemented & the nib files (.xib file) designed for this version of the application are summarized below.

WeatherViewController (Figure 21)

This is the first view to appear when the application starts. This view displays the weather forecasts (for the device's current time) and lets the user add the place as a favorite (with a star in the top right corner) and also pushes another view to the navigation controller when a user taps a metric.

MetricViewController (Figure 22)

This is the view that is pushed inside the navigation controller stack of 'WeatherViewController' every time the user clicks a metric. It is transparently designed (which means that it is used to display any of the seven metrics) and it contains controls to switch to forecasts by day, instead of the default view, by hour. It also has a button to display the metric values as a chart.

Figure 21: WeatherViewController screenshot – version 1.0

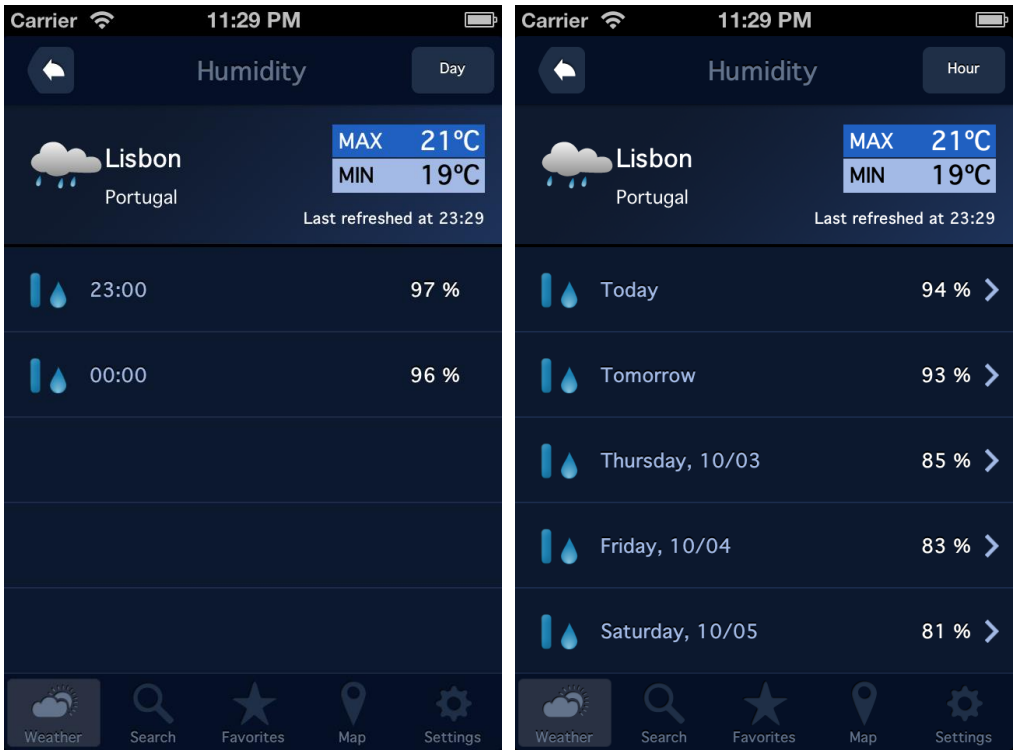

Figure 22: MetricViewController screenshot – Hour and day view

GraphViewController (Figure 23)

This view appears when a user clicks on any of the day's forecasts. It presents a graph showing the values from 00:00 to 23:00 of the selected day.

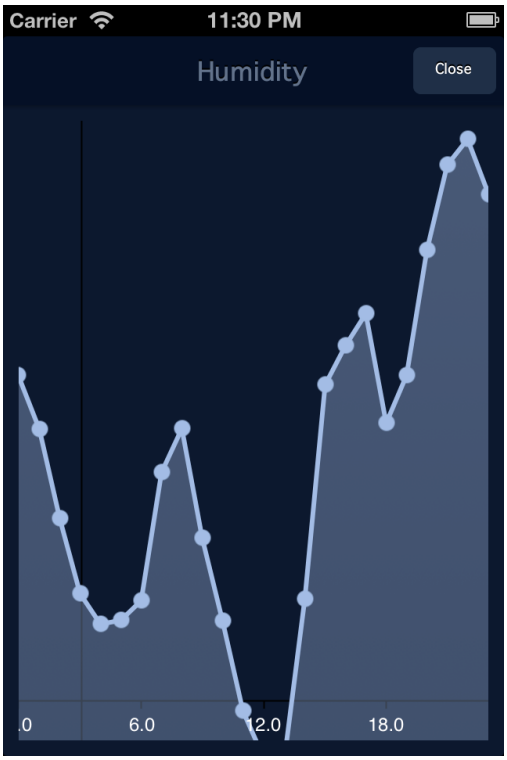

Figure 23: GraphViewController screenshot

SearchViewController (Figure 24)

This view is used to search for a place using the Google Places API. For every character typed in the text field, a request is made to the API with new results, and the list refreshes its options. When the user clicks on an item, the app sends him to tab one ('WeatherViewController') with the forecasts for the chosen location.

FavoritesViewController (Figure 25)

This view is used to display the favorite places the user has saved and it provides a quick way to delete them by clicking the 'Edit' button in the top right corner of the view. When an item from the list is tapped, the app sends the user to tab one with the forecasts for the chosen favorite.

MapViewController (Figure 26)

This view is used to display the favorite places as pins drawn in a map, and the user can simply click one pin to get information about the favorite and click it again to go to the tab one with the forecasts.

SettingsViewController (Figure 27)

This view is used to display the controls that let the user change the units of temperature and wind velocity. Both the settings and the favorites are stored locally, in a file, as dictionaries (key valued pairs).

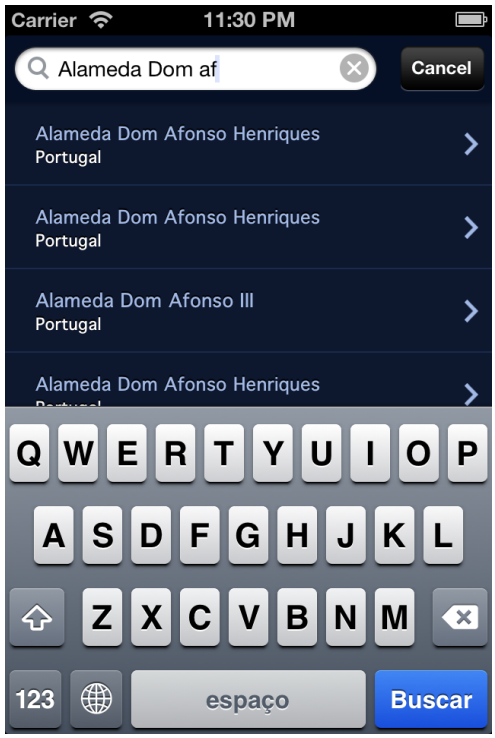

Figure 24: SearchViewController screenshot

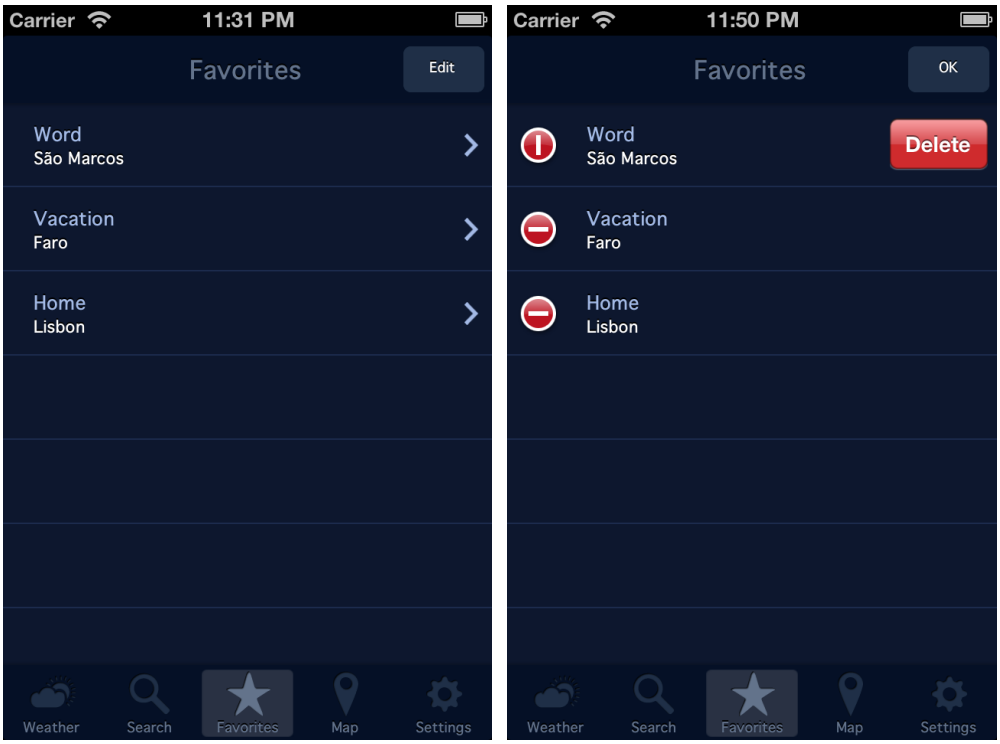

Figure 25: FavoritesViewController screenshot

Figure 26: MapViewController screenshot

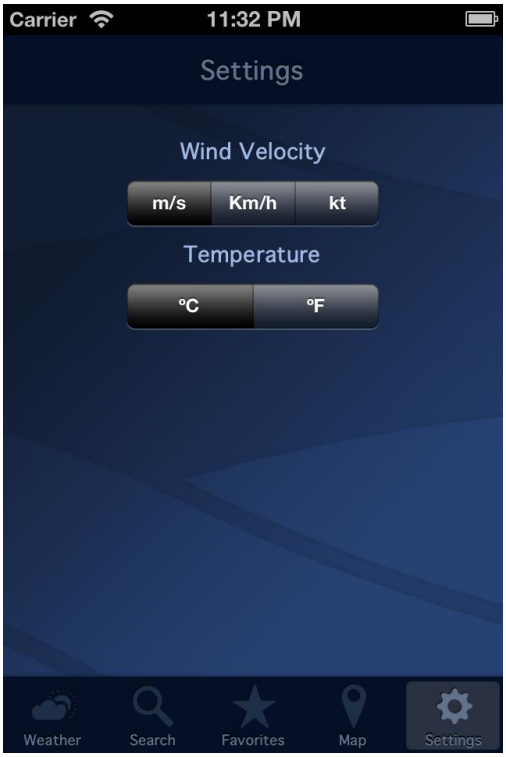

Figure 27: SettingsViewController screenshot

4.1.8 Problems encountered

4.1.8.1 Coding in Objective-C

My learning curve for Objective-C was steeper than I expected. Having had academic experience with C, I thought it could not be that different since it is a variation of this language. Objective-C is an object-oriented programming language but on top of that, it adds Smalltalk-style messaging to the C programming language. Smalltalk is also a dynamically typed, object-oriented language. It is a reflective programming language that was created in 1970 for educational use with the aim to underpin the "new world" of computing. In summary, Objective-C is easy to read but hard to write. It is easy to learn how to create a method in Objective-C, but because of all the semantics, it takes a while for a developer to fully understand it.

4.1.8.2 METEO-IST's services were slow

Another problem we faced while developing the app was that the METEO-IST's services were taking too long to respond. This was because every time a service was called, it would access the forecast data file which had more than 1 Gigabyte, and get the information it required. Because that file contained more data then what the app needed (i.e. it had temperature in various layers of the atmosphere), Rosa Trancoso created a separate file with only the information the app required. The file became smaller than 300 Megabytes and the services got faster.

4.1.9 Testing

In order to transfer the app to users, for testing, a physical connection with the device was required. When the app was finished, a meeting was scheduled with some prospective users so I could connect their devices to the Mac and install the app. One by one, the app was installed and some days later I started to receive feedback.

One of the bugs people were reporting was that their favorites would be wiped some days after they added them. It turned out that I was storing the favorites and settings in temporary files, which were managed by iOS and were deleted when the operating system decided to do so. I solved this issue by using permanent files that are only managed by the application.

4.1.10 Release

After I had finished resolving all the bugs I submitted the app to the App Store on April 10, 2013. It took 7 days for Apple to approve my application and release it on the App Store. It usually takes Apple about 2 or 3 hours to review an application (depending on the app features) so the remaining time was just waiting for someone to review it, probably because of the hundreds of applications that are submitted every day.

The app was released to the public on April 17 and an announcement in IST Facebook page, Figure 28.

Figure 28: The 'Weather IST' release announcement on IST official Facebook page

4.1.11 User satisfaction

The application took about 5 months to develop, test and integrate with METEO-IST. We were eager to get results and see how the application was doing on the market. The first results were good: the app had about 500 downloads in the first two weeks and was able to maintain its rank position in the top 40 (in a total of 200 weather apps). Figure 29 and Figure 30 show the evolution of downloads and rank (by week) of the app in the App Store in its lifetime.

The app also received some good reviews and was able to score a 4 star rating out of 5 from users.

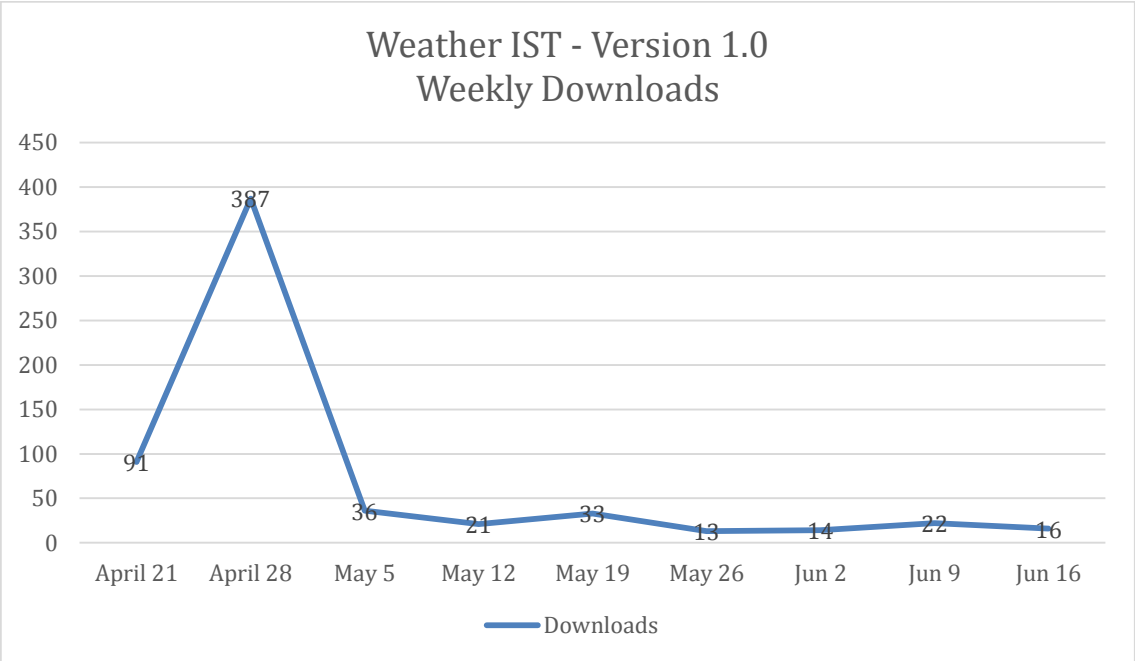

Figure 29: ‘Weather IST’ - Version 1.0 downloads

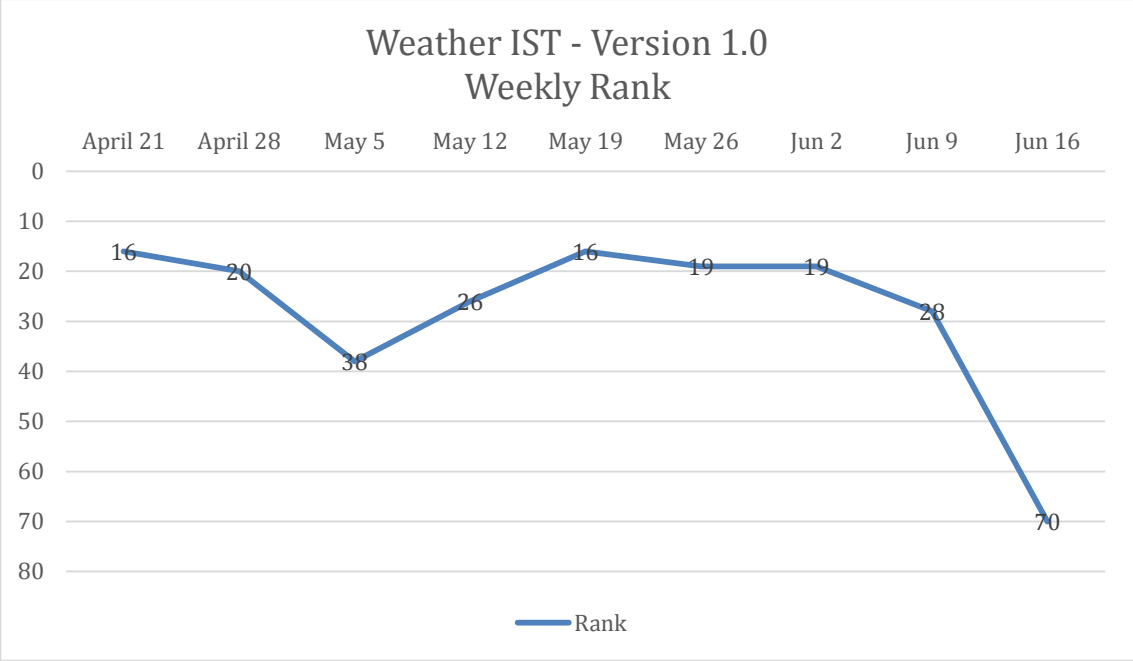

Figure 30: ‘Weather IST’ – Version 1.0 rank

4.2 Version 1.0.1

Some months before the first version of ‘Weather IST’ was launched, Apple released its new iPhone 5. Throughout the years Apple had launch the device with the same screen size (3.5 inches), but with the iPhone 5 they decided to go bigger and increased the screen size to 4 inches. The screen only grew in height, not in width. This changed the resolution from 960 x 640 pixels to 1136 x 640 pixels. Developers now had 176 more pixels to draw on their app. This change came with the release of the iOS 6 and consequently brought a new SDK for developers with Xcode 4.

For apps that were not ready to take advantage of the iPhone 5 screens, iOS would put black bars on top and bottom of the screen to fill the remaining space displaying the app with the same height as it was initially developed. Our app was one of those cases as it was still compiled in the previous SDK and did not reflect these changes. I then recompiled the application and made some adjustments to the interface. It required some tweaking but overall it was fairly simple to recompile the app and take advantage of the 4-inch screen. Figure 31 shows the height difference in ‘Weather IST’ on an iPhone 5 and an iPhone 4S.

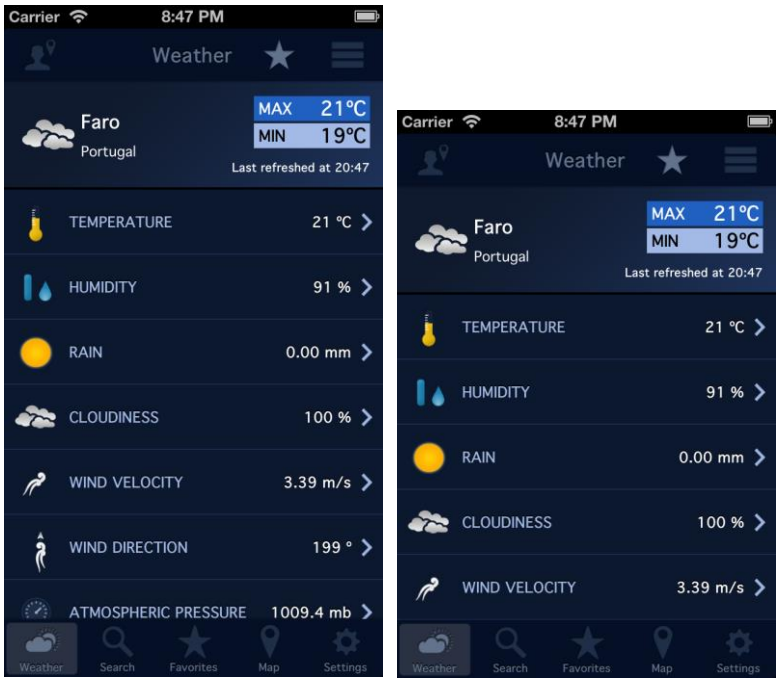

Figure 31: 'Weather IST' version 1.0 in iPhone 5 and iPhone 4S

I was also not obtaining as many reviews as I wanted, so I took the opportunity to implement a rate reminder system in the app. There are some packages already developed to address this and by searching the Internet I found a very popular one named 'iRate'. 'iRate' is a library that helps promoting iPhone (and Mac) apps by prompting users to rate the app after using it for a few days. This approach is a very good way to get positive app reviews by targeting only regular users (who presumably like the app or they would not keep using it). I configured the library to remind the user to rate the app every 5 days, unless he had rated it already or chosen the 'No Thanks' option since the last thing I wanted is for a user to stop using the application because it is annoying him with rating requests.

4.2.1 User Satisfaction

I submitted the app on June 15, and it was released on the App Store on June 23, 9 weeks after the launch of the first 'Weather IST' version. Apple's analytics dashboard separates downloads from updates. So we can assume a download is a new user (or one that deleted the previous version) and an update is a current user that just updated his previous version.

Based on this fact, the release of version 1.0.1 showed me that 'Weather IST' had about 200 active users (that used the app on a regular daily basis) because in the first 2 weeks the app had more than 200 updates. As expected, the number of new downloads did not change much compared to the previous version because the app did not have anything new to offer to users. Figure 32 and Figure 33 show the evolution of downloads and rank (by week) of the app in the App Store in its lifetime.

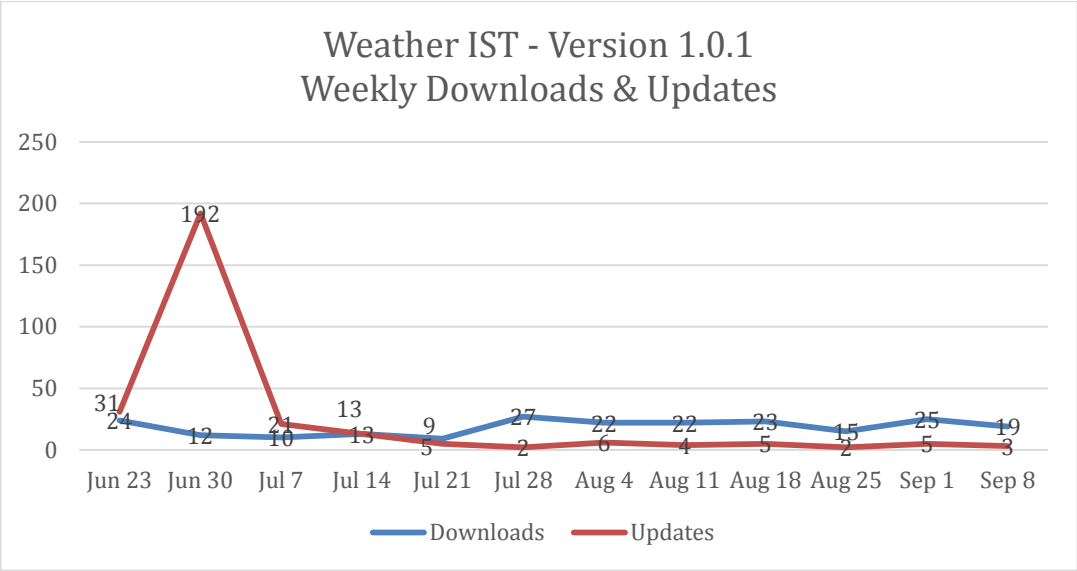

Figure 32: 'Weather IST' - Version 1.0.1 downloads

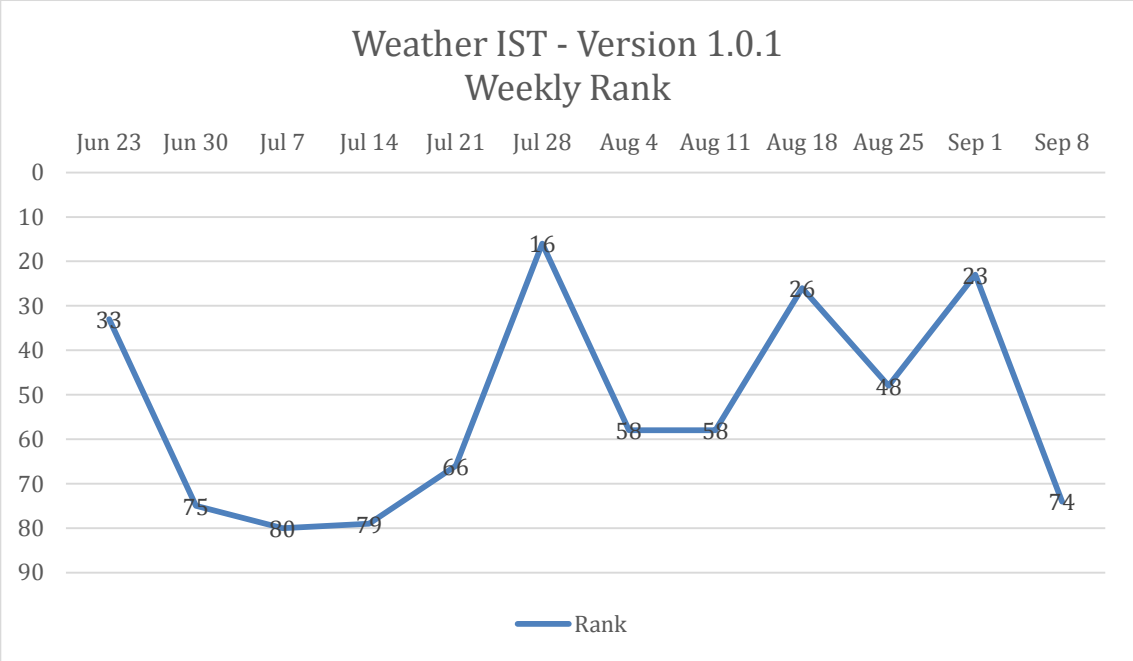

Figure 33: 'Weather IST' - Version 1.0.1 rank

4.3 Version 2.0.0

As I was writing this report I was not really happy with the application's final result. While using it, it felt complicated, and the way the information was shown was not taking advantage of the powerful system supporting it (METEO-IST). I started writing down some issues that had to be addressed.

How can the user check the forecasts without having to tap the application so much and push so many views? How can a user check if the next day will be a clear and worm day? After he chooses a location, he has to tap the rain metric, then tap on 'Day' and see the value for tomorrow. Then he has to go back, tap the cloudiness metric, tap the 'Day' button again and check if it is at 0%. If he is also

worried about the temperature, he has to go back again and tap the temperature metric, then tap ‘Day’ and check the maximum and minimum for tomorrow. Was this easy? No. Now imagine the question above only this time the user wants this information for a specific hour of the day. For each metric, and after switching to the day view, he has to click the day to open up the graph and check the value for the hour he wants. What is the difference between favorites view and map view? In terms of functionality, none.

I started rethinking the application.

4.3.1 Requirement Analysis

Before I go further I wanted to recall a quote from Steve Jobs.

“Simple can be harder than complex. You have to work hard to get your thinking clean to make it simple.”

Personally I think he was absolutely right. Doing something simple that is easy to use but at the same time responds to the needs of the user, is much harder than doing something complex that has perhaps more features but frustrates the user to the point where he will just stop using it.
With this thinking in mind, I wrote down the changes I wanted to make for this new version.

- One view only for all the weather information. No more navigating through views, or opening graphs. The user would have all the information in just one view.
- Tab bar removed. If I want to have all the information in one screen we have to remove elements that only cause noise. Elements have to breathe in order for this design to work.
- The map view brings no value to the app, so it should be removed.
- A view just for the favorites is also too much. With this new design I wanted the user to access the favorites without having to push another view.
- The app has to become lighter. The first version of ‘Weather IST’ was too dark and plain. Users care much about design and they can easily switch to another application even if its forecasts are not as good. I chose to implement dynamic backgrounds that reflect the current weather.
- Weather icons are really good because they aggregate the time of day (sun or moon), cloudiness and rain. So if we want a cleaner design we have to start using more of these.

After I wrote down these changes, I began drawing a prototype. The new navigation was based on Facebook’s app for the iPhone. Figure 34 shows the interface I planned for this new version. By clicking on the top left button, the menu would come from the left side with all the app options like ‘Local Weather’ which tries to get the user’s location, ‘Search Location’ which opens the same view it did in the first version, the user’s favorite places (without the need for another view) and the ‘Settings’ button which opens the settings view. Also I created social pages, both in Twitter and in Facebook, for the app, so users had a place where they could get information about future releases, known bugs and even any app downtime due to the METEO-IST servers. It features the ‘Rate this app’ button which takes the user

to the app page on the App Store so he can rate it, and also the IST logo which when tapped, would redirect the user to the college's web page.

4.3.2 Use Case Analysis

4.3.2.1 Forecasts for the next hours

In the previous version a user could only check the forecast for the next few hours for the present day. For example, if he was checking the temperature forecasts at 22:00 he would only see the values for the current hour and for 23:00. He did not have a way of checking the hourly periods after that and his only option was to change to day view and see the maximum and minimum temperature for the next day. In this version the user is able to check the forecast for the next 24 hours regardless of the time of day he is requesting them at. He will even be able to check the hourly forecast for any of the next following 7 days for any of the metrics.

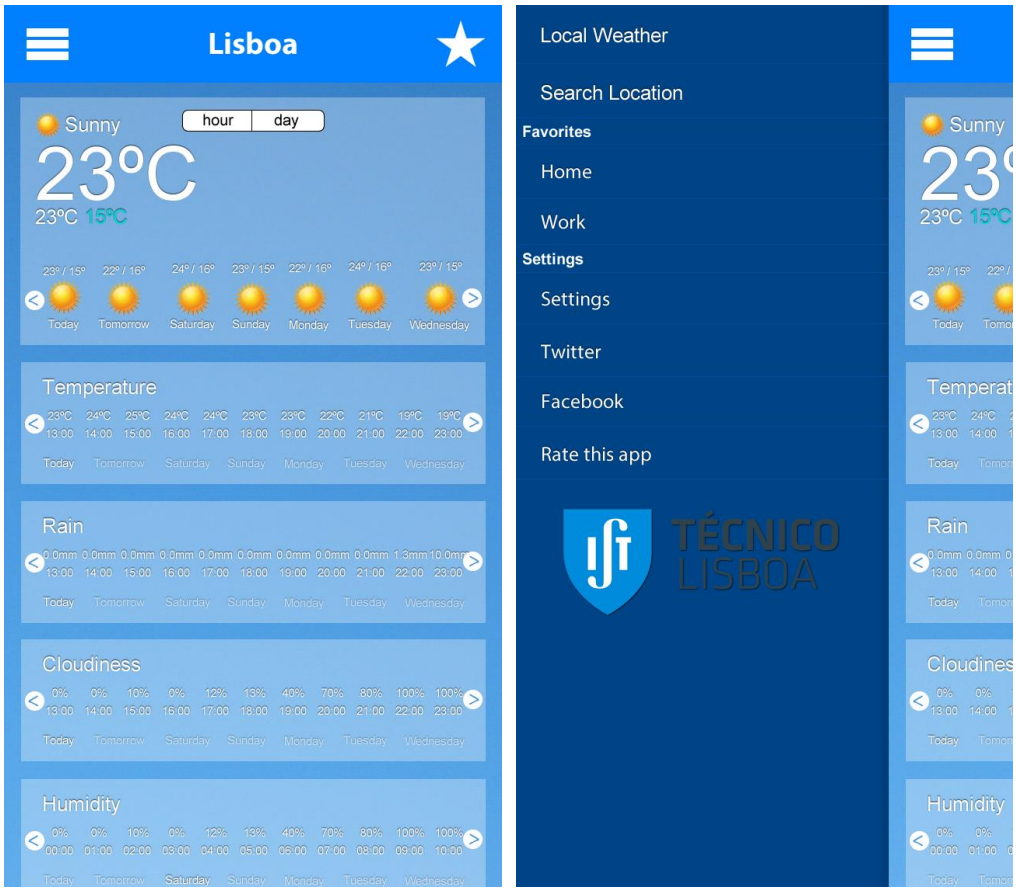

Figure 34: 'Weather IST' version 2.0.0 prototype

Description: The user wants to check the cloudiness forecast for the following hours on a favorite he has named 'Home'.

Precondition: The user has added, at least, one location as a favorite.

1	The user taps the top left button.
2	The app scrolls a menu from the left.
3	The user taps on a favorite called ‘Home’.
4	The app closes the menu and refreshes the forecasts for the chosen favorite.
5	The user scrolls the view until the cloudiness cell appears and views the results.

Table 20: Forecasts for the following hours – Main Success Scenario

4.3.2.2 Remove a favorite place

The favorites view was removed from this version. So the way a user deletes a favorite is the same way he adds it, which is by clicking the top right corner star button.

Description: The user wants to remove one of his favorite places.
Precondition: The user has added, at least, one location as a favorite.

1	The user taps the top left button.
2	The app scrolls a menu from the left.
3	The user taps on the favorite he wants to remove.
4	The app closes the menu and refreshes the forecasts for the chosen favorite.
5	The user taps on the star button on the top right corner of the screen.
6	The app shows an alert asking the user if he really wants to delete the favorite.
7	The user taps on ‘YES’
8	The app removes the favorite.

Table 21: Remove a favorite place - Main Success Scenario

4.3.3 METEO-IST’s Web Services

4.3.3.1 PrevForNextHoursByMetric.php

To support the new Use Case, described in section 4.3.2.1, we developed a new service that given a date (with time), latitude, longitude and a metric returns the next 24 hours of forecast for that metric beginning at the given time. The service was developed by Carlos Palma who replaced Rosa Trancoso as a supporter of the METEO-IST when she left for another position.

Parameter name	
Lon	Longitude (float with four decimal places)
Lat	Latitude (float with four decimal places)
Date	Date (string with the format yyyyMMdd_HH)
Metric	Metric name (string from the enumerator)
Example Call	
HTTP	PrevForNextHoursByMetric.php?lon=8.9341&lat=38.1294&metric=temp&date=20120813_15
Response example	
JSON	{ response : { points : [15, 14, ... , 16] } }

Table 22: PrevForNextHoursByMetric.php – Service Interface

The response has a root object named ‘response’ and its value is an array of exactly 24 values that correspond to the 24 hours after the requested time.

4.3.4 Development

I created a branch from version 1.0.1 and started developing from that point. As I removed the tab controller I needed to implement the new app navigation, which was the left menu opening when the top left button was clicked. As I stated above, I got this idea after using the Facebook app and enjoying the experience, and while browsing on the Internet I found that someone had already created a library to mimic its navigation. The library is called ‘IIViewDeckController’ and it is freely available even for commercial uses. It is very easy to use and works just like any native navigation controller.

I also needed a control that would let a user scroll the metrics horizontally. The SDK natively supports this feature by allowing the developer to use a ‘UIScrollView’. The ‘UIScrollView’ class provides support for displaying content that is larger than the size of the application’s window. It enables users to scroll within that content by making swiping gestures, and to zoom in and out from portions of the content by making pinching gestures. The pinching gesture and zoom are present to support the interaction with the images and by putting an image inside a scroll view, it allows the user all these actions. In my case I did not need any zooming or pinching I just needed to swipe left and right to navigate through the elements. When using this control, all the content inside it has to be drawn programmatically and in the end of the iteration we have to tell the control how much width the content inside of it has. That is what tells the scroll view its bounds for scrolling. For example, as I do not need any vertical scroll in these views, I just tell the scroll view that it’s content size is equal to its real size in the screen therefore disabling vertical scrolling.

4.3.5 Testing & Release

I performed the only testing work on this version as I did not distribute it to anyone including my supervisor. The reason for that is that I wanted to have this version finished in time to present it on this

report as well as collect the results for user satisfaction. Skipping this step allowed me to jump directly to release. The app was submitted to the App Store in September 3, and was approved and released by Apple on September 11.

4.3.6 User satisfaction

To compare this version with the previous two, I had to change some variables for the comparison to be fair. I will be merging version 1.0.0 and 1.0.1 into one (1.0.x) and summing their statistics. The reason I am doing this is because both versions are practically the same for the regular user. By this I mean that although a user may not have liked version 1.0.0, and version 1.0.1 by extension, he could however like version 2.0.0 as it is completely different.

Also, although the charts for the previous apps presented the number of downloads/updates per week, this version has been on App Store for only 3 weeks so I decided to also include a chart with downloads/updates per day to get a better notion of the difference between them.

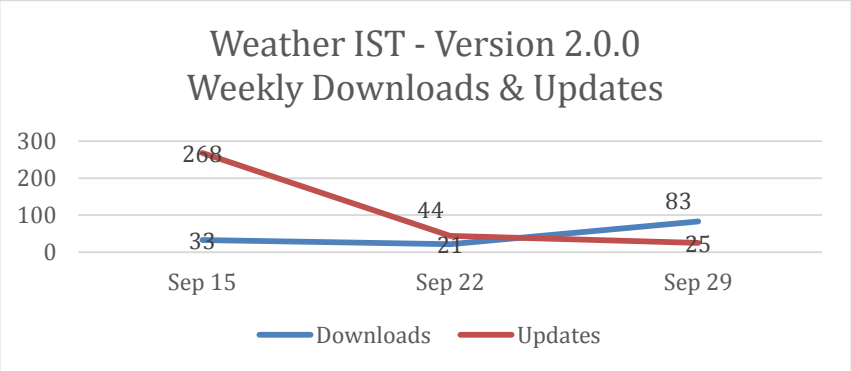

Figure 35: 'Weather IST' - Version 2.0.0 downloads/updates per week

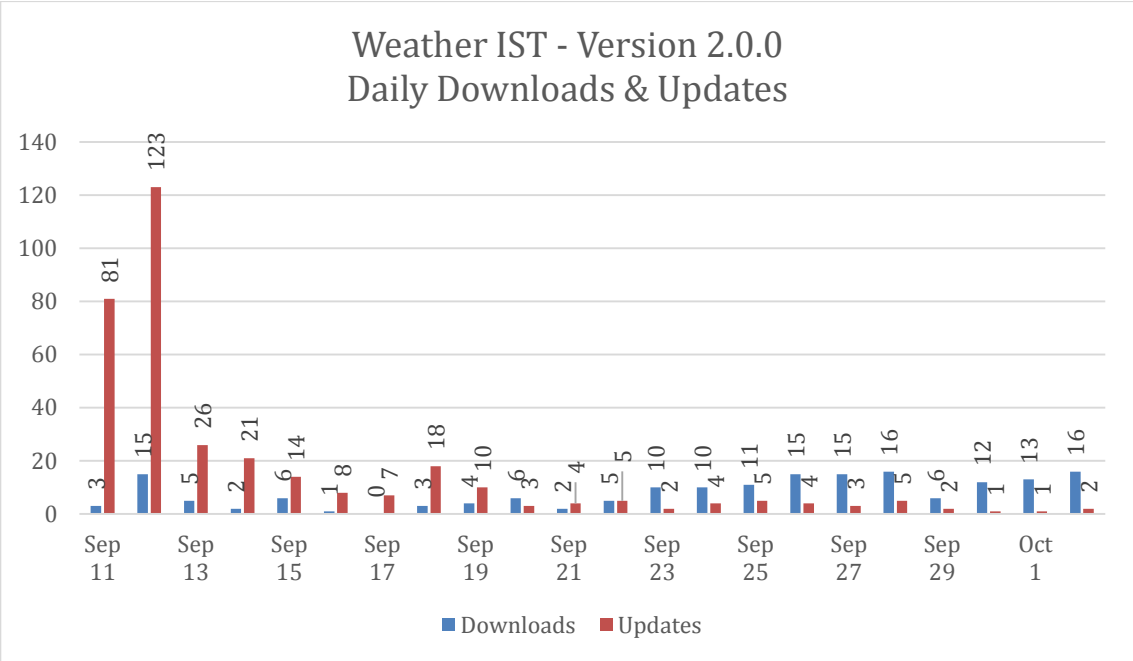

Figure 36: 'Weather IST' – Version 2.0.0 downloads/updates per day

Figure 35 shows that version 2.0.0 had more than 300 updates in the first 3 weeks, which is 50 more than version 1.0.x for the same period after release. On the first 2 days alone there were more than 200 updates (Figure 36), which shows that users were eager to upgrade the app. Also, the same chart shows an increase in downloads beginning on September 23, which coincided with the beginning of autumn and the first rainy days in Portugal. The third week of version 2.0.0 had more downloads than several weeks of version 1.0.x put together. Figure 37 shows a graph that compares version 2.0.0 and version 1.0.x in terms of downloads per day. This metric allows us to compare both versions fairly, seen as version 2.0.0 has been on the App Store only for three weeks and version 1.0.x for almost five months, so an overall comparison would not give any interesting results.

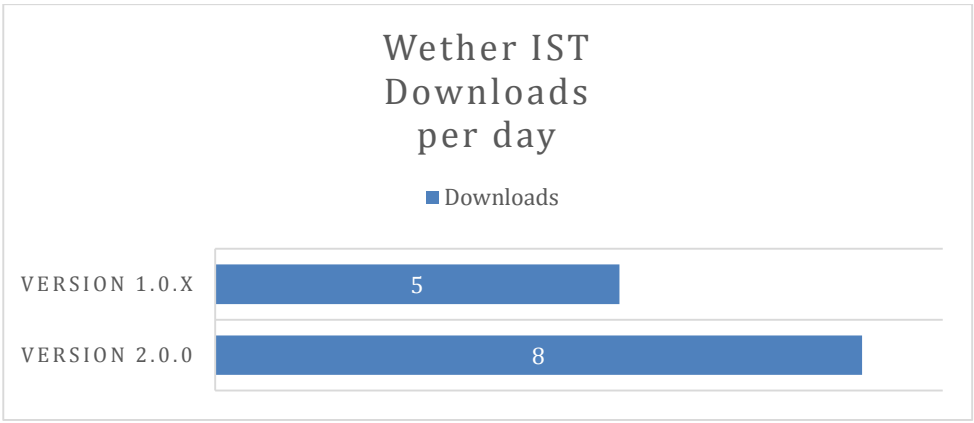

Figure 37: 'Weather IST' – Version comparison by downloads per day

Although Figure 37 shows that the number of downloads per day for version 2.0.0 is higher than for version 1.0.x it does not emphasize the real difference. When an app is released on App Store for

the first time, and not as an update, it gets featured on the new apps section on the store. This tends to give the app a boost in downloads in the first few days which then tend to normalize depending on the popularity and quality of the app. That is the reason why there was no boost in downloads on the days following the launch of both 1.0.1 and 2.0.0 versions. If we discard the boost in downloads on the first few days of version 1.0.0 because they do not give a realistic sense of the app's popularity, but only that it was new, we end up with a chart, shown on Figure 38, which provides a more realistic image of the differences between the two major versions.

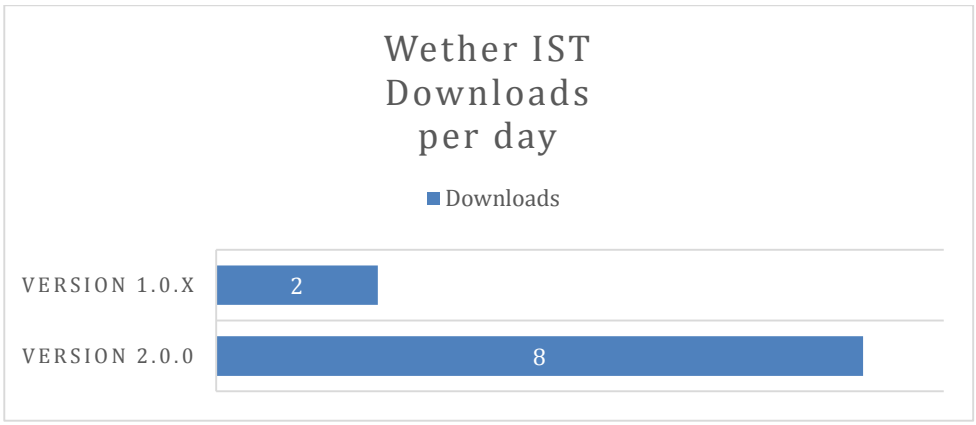

Figure 38: 'Weather IST' – Version comparison by downloads per day not counting the app's launch

What this chart tells us is that after the initial launch, the first major version of 'Weather IST' was only gaining 2 new users every day, whereas 2.0.0 version is gaining 8 new users every day. It is a 400% increase compared to its predecessor and the next chart (Figure 39) shows the impact of this change in months.

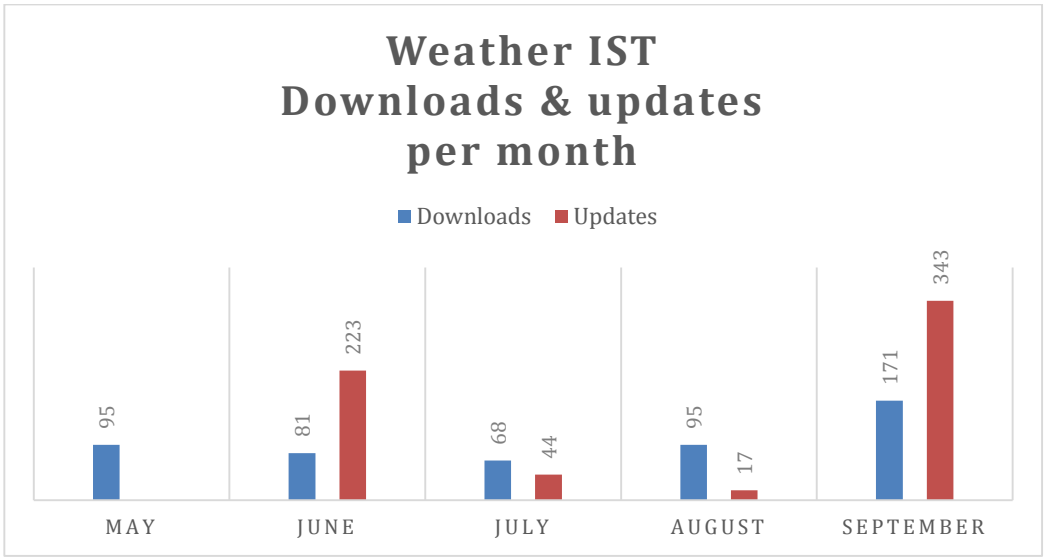

Figure 39: 'Weather IST' – Overall downloads and updates per month

To draw this chart I removed the first month of downloads because it does not give a realistic sense of the app's popularity. Notice that even though September has 180% more downloads than the previous month, the 2.0.0 version only came out on September 11 which means that the actual increase could be higher. One good comparison can be made when all the statistics from October are available. Also notice the difference in updates. In June, when version 1.0.1 came out, it had 223 updates, whereas when version 2.0.0 came out in September it had almost 350 updates.

The ranking of version 2.0.0 has also been better than previous versions. Again because the time span is short compared to the previous versions, I decided to show this next chart (Figure 40) in days rather than weeks. The chart shows stable values beginning on September 23 which coincide with the period of increased number of downloads shown in the chart in Figure 36.

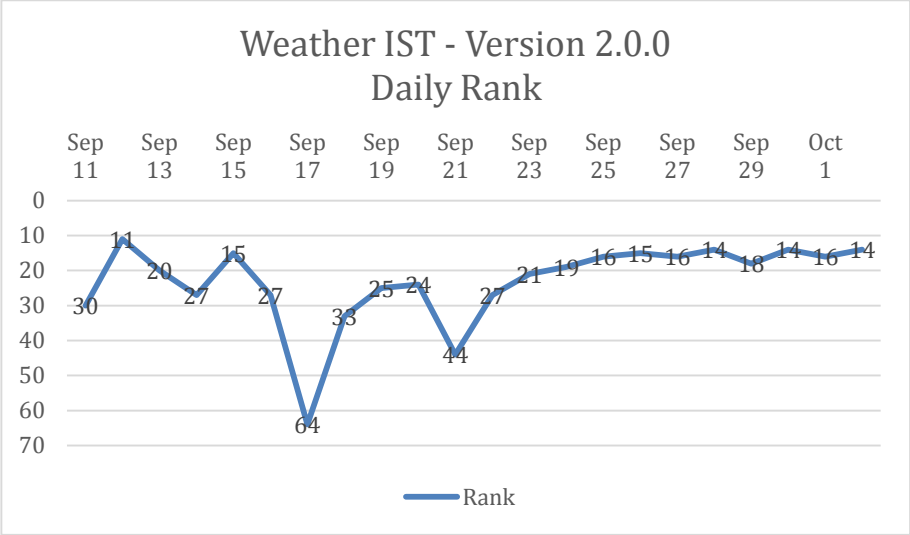

Figure 40: 'Weather IST' – Version 2.0.0 daily rank

The application has now more than 1000 downloads. It surpassed that number on September 1 with most of downloads being made in the Portuguese market although there were some in other countries as well. The following chart, in Figure 41, shows the top 5 countries overall.

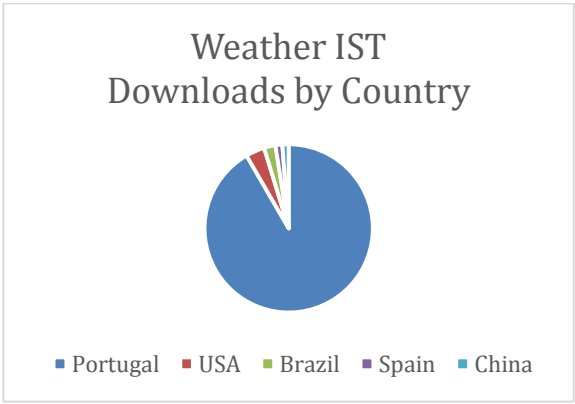

Figure 41: 'Weather IST' – Overall downloads by country

5 Conclusions

I am really happy with the final result of 'Weather IST'. The application got a more professional look and feel, and it is also very easy to use compared to the previous version. Figure 42 shows the app new design.

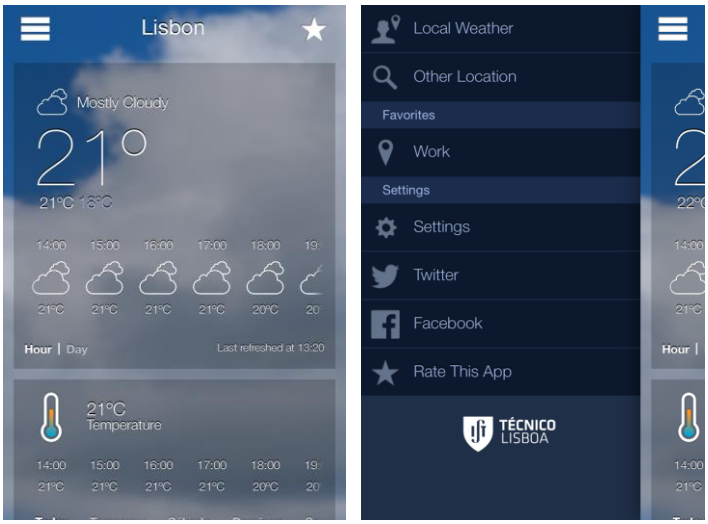

Figure 42: 'Weather IST' – Version 2.0.0

Mobile app development is a continuous process with many iterations. There is no such thing as a perfect design and the app has to dynamically be able to change in order to follow the trends, because the App Store is a trend market. What is good today, may not be good tomorrow. Developers have to always keep this in mind if they want their app to succeed in this market, and although testing is a very important step of this process, so the app is not published with bugs, the best test they can make to it is to release it on the App Store. It will reach millions of users throughout the world, all with different needs and tastes and only then will you have the results that allow you to develop the app in one or more directions.

5.1 Things learned

5.1.1 Design

Steve Jobs once said: "Design is not just what it looks like and feels like. Design is how it works." I tend to agree with this. In the first version of 'Weather IST' I misjudged this statement. I thought that by having great and precise forecasts the design would just be a way of showing those forecasts. It is not. The design has to reflect the quality of those forecasts because the user has no idea of what the services can or cannot do. He only knows what he sees and interacts with.

5.1.2 Objective-C

Objective C was a completely new experience for me. All other object-oriented programming is relatively easy since the syntax is simple. All the brackets and the pointers with a different syntax for function call,

memory management etc. offers a level of resistance at the beginning. I overcame this difficulty using some materials available on the web.

5.1.3 Xcode and Interface builder

The IDE was new, as well as the development and debugging techniques. The graphic tool, Interface builder used to build the user interface seemed initially difficult, and some features were different from what I was used to, like connecting the controls and attaching events on the builder file to the view controller file.

5.1.4 App Store market

App Store is like any market out there. It is not just about developing a great application, it is also about good marketing. One thing I learned is that, if you are about to release an app that will only have forecasts for a specific country, do not release it in the summertime. People will simply not use it. The tendency to open a weather application during the summer is too low compared to other periods in the year. The probability of having 2 or 3 months of sun and clear sky is very high in the summer and because of that, users will not be searching for these.

5.2 Future Work and Improvements

The following sections describe the improvements that could be done.

5.2.1 Forecasts for the entire world

Although the application only serves forecasts for Continental Portugal, it is available to download on the App Store in the entire world. Even if users in other countries like the application they cannot really use it in their places. What should be done is to open the search location to any country and if the chosen location is outside of Portugal, use an external service to get the weather forecasts while displaying them in the same way as they do now. There are a number of weather API available in the market like weather.com or wunderground.com.

5.2.2 More metrics

One metric that most apps have is sea wave height. Surfers love it as it tells them where to go. Also METEO-IST's new website shows a new metric which is the temperature felt. This temperature is different from the one that the application is showing at the moment since it considers wind and other factors. So I could implement it on the app and have something like: current temperature is 22ºC but in reality it feels like 23ºC.

5.2.3 iOS 7

A few days after the second version of 'Weather IST' came out, the new version of iOS (7) was released. 'Weather IST' has not been compiled (optimized) for the new SDK yet, although it works fine on iOS 7. One future task will be to recompile using the new operating system and resolve any issues that may arise.

5.2.4 Forecasts charts

Although most users just prefer to see the forecasts in a quick way with icons or text, some liked the charts. One future improvement would be to bring back the charts to the application.

6 Bibliography

[1] Allan, A. (2012). *Learning iOS Programming, Second Edition.* Sebastopol: O'Reilly.

[2] Apple. (2008, March 12). *iPhone SDK Downloads Top 100,000.* Retrieved September 21, 2013, from Apple: http://www.apple.com/pr/library/2008/03/12iPhone-SDK-Downloads-Top-100-000.html

[3] Apple. (2009, January 21). *Apple Reports First Quarter Results.* Retrieved September 21, 2013, from Apple: http://www.apple.com/pr/library/2009/01/21Apple-Reports-First-Quarter-Results.html

[4] Apple Inc. (2012, June 11). *Nib Files*. Retrieved from Mac Developer Library: https://developer.apple.com/library/mac/documentation/Cocoa/Conceptual/LoadingResources/CocoaNibs/CocoaNibs.html

[5] Apple Inc. (2013, September 18). *About the iOS Technologies*. Retrieved from iOS Developer Library: https://developer.apple.com/library/ios/documentation/Miscellaneous/Conceptual/iPhoneOSTechOverview/Introduction/Introduction.html

[6] Apple Inc. (2013, September 18). *iOS Simulator User Guide*. Retrieved from iOS Developer Library: https://developer.apple.com/library/ios/documentation/IDEs/Conceptual/iOS_Simulator_Guide/Introduction/Introduction.html

[7] Isaacson, W. (2011). *Steve Jobs: The Exclusive Biography.* Los Altos: Simon & Schuster.

[8] Lynch, P. (2007, March). The origins of computer weather prediction and climate modeling. *Journal of Computational Physics 227*, 3431-3444.

[9] Trancoso, A. R. (2012, April). Operational Modelling as a Tool in Wind Power Forecasts and Meteorological Warnings.

7 Appendix

7.1 Available weather application features

	Weather Live	Weather Genie	Weather HD	Termomether!	AccuWeather	The Weather Channel
Price	€1,99	€1,99	€1,99	Free	Free	Free
Reviews Number	21.000	400	17.000	300	83.000	360.000
Average Rating	4.5	4	4	4	3.5	3.5
Include Publicity	No	No	No	Yes	Yes	Yes
24 hour forecasts	Yes	Yes	Yes	Yes	Yes	Yes
Number of day with forecasts	7	7	7	1	15	10
Temperature	Yes	Yes	Yes	Yes	Yes	Yes
Cloudiness	Yes	Yes	Yes	No	Yes	Yes
Rain	Yes	Yes	Yes	Yes	Yes	Yes
Atmospheric Pressure	Yes	Yes	Yes	Yes	Yes	Yes
Humidity	Yes	Yes	Yes	No	Yes	Yes
Wind speed & direction	Yes	Yes	Yes	Yes	Yes	Yes
Visibility	Yes	No	Yes	No	Yes	Yes
Social Network Sharing	No	No	Yes	No	Yes	Yes
iCloud	No	No	No	No	Yes	No
Allows unit changing	Yes	Yes	Yes	No	Yes	Yes

7.2 Provisioning the device for development

To install the app on the device, a developer needs go through some steps as depicted in the figure below.

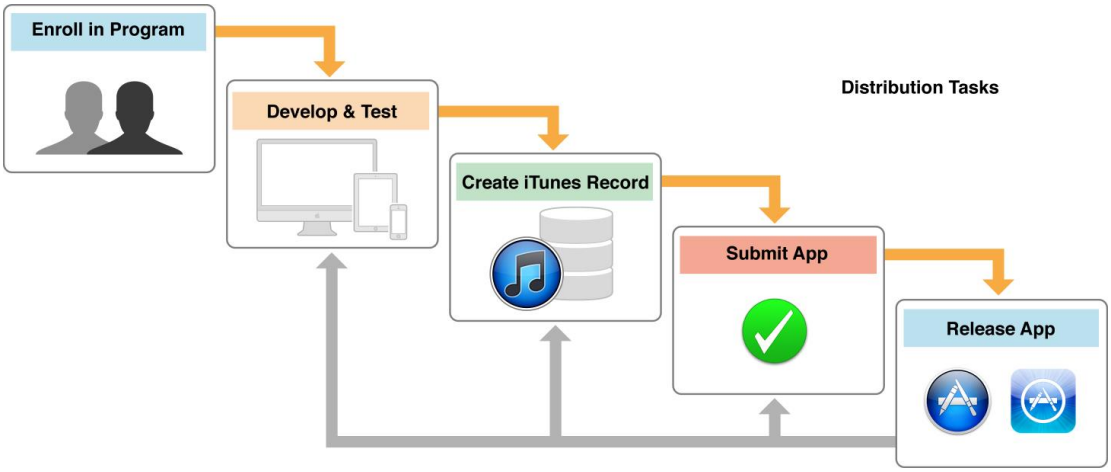

Figure 43: App Distribution Workflow

Some of the key steps are summarized below:

7.2.1 iOS Development Certificate

The developer should create a CSR (Certificate Signing Request) using the Certificate Assistant on the Mac. After this the developer needs to login to the Apple's member center. In the provisioning portal on the Certificates -> Development tab, using the add certificate button to upload the CSR. This awaits approval from the team admin who needs to sign the CSR. The admin signs it by going it to the Development tab of certificates section.

7.2.2 Installing & downloading iOS Development Certificate

In the Certificates -> Distribution section the WWDR Intermediate certificate is downloaded and installed on the machine through Keychain Access.

7.2.3 Adding Devices to your team

By adding a device's UDID (Unique Device Identifier) into the provisioning profile, the app can be installed on the device for testing. Around 100 devices can be added on a development team. The UDID can be found to the iTunes Summary tab when the device is connected. It will be a 40 character string.

7.2.4 iOS Provisioning Profile

For development or distribution, a provisioning profile needs to be created. This can be either a development iOS provisioning profile or an iOS distribution provisioning profile.

7.2.5 Build & Distribute

The application is built and code signed through Xcode and can be distributed to the app store or an Ad Hoc distribution.

Each of the steps is explained in detail in the iOS Provisioning Profile on the Apple Developer's Website.