University Institute of Lisbon

Department of Information Science and Technology

Design and development of a hybrid-based mobile app for ISCTE-IUL

Fábio Paulino

A Dissertation presented in partial fulfillment of the Requirements
for the Degree of
Master in Computer Science

Supervisor
Dr. António Luís Lopes
ISCTE-IUL

September 2015

Resumo

O aumento do poder da computação móvel e a quantidade de dispositivos móveis em todo o mundo levou a um considerável aumento de aplicações. Muitas destas aplicações móveis foram criadas para ajudar os seus utilizadores no seu dia-a-dia. Algumas universidades viram este aumento como uma oportunidade para facilitarem a vida dos seus alunos ao criarem aplicações para simplificar o acesso aos diversos serviços.

No ISCTE-IUL, a descentralização dos diversos serviços e a falta de compatibilidade com dispositivos móveis dificultam o acesso destes serviços aos seus alunos, os quais necessitam de aceder aos mesmos através dos seus portáteis ou computadores da instituição.

A proposta apresentada nesta dissertação passa pela criação de uma aplicação móvel utilizando tecnologias híbridas de forma a disponibilizar um meio mais fácil de interação entre os alunos e os serviços do ISCTE-IUL.

Os resultados obtidos com os alunos da instituição mostram que esta aplicação é viável e que é desejável pela comunidade estudantil.

Palavras-chave: Aplicação Móvel, ISCTE-IUL, Aplicação Híbrida, API

Abstract

The increase in mobile computing power and the number of mobile devices available worldwide has led to a considerable increase in applications. Many of these mobile applications are designed to help its users in their daily basis. Some universities saw this increase as an opportunity to help their students in their academic life by creating a mobile application that grants easier access to several services of the institution.

In ISCTE-IUL, the decentralization of several services and the lack of compatibility with mobile devices hinder their access to their students which then they require laptops or computers of the institution to access them.

The proposal presented in this dissertation is the creation of a mobile application using hybrid technologies in order to provide an easy mean of interaction between students and services of ISCTE-IUL.

The results obtained with the students of the institution show that this application is viable and that it is desired by the student community.

Keywords: Mobile Application, ISCTE-IUL, Hybrid Application, API

Contents

List of Figures

Abbreviations

ISCTE-IUL	**ISCTE** - **I**nstituto **U**niversitário **L**isboa
API	**I**nterface de **P**rogramação de **A**plicações
Oxford	University of **Oxford**
Harvard	**Harvard** University
UIS	**U**niversity of **I**llinois **S**pringfield
UA	**U**niversidade de **A**veiro
UCP	**U**niversidade **C**atólica
Lusófona	**U**niversidade **L**usófona
IST	**I**nstituto **S**uperior **T**écnico
OS	**O**perative **S**ystem
MVC	**M**odel-**V**iew-**C**ontroller
SOAP	**S**imple **O**bject **A**ccess **P**rotocol
WSDL	**W**eb **S**ervice **D**efinition **L**anguage
PHP	**P**HP: **H**ypertext **P**reprocessor
POI	**P**oint **O**f **I**nterest
LDAP	**L**ightweight **D**irectory **A**ccess **P**rotocol
UML	**U**nified **M**odeling **L**anguage
SQL	**S**tructured **Q**uery **L**anguage
DTO	**D**ata **T**ransfer **O**bject
PDO	**P**HP **D**ata **O**bjects
CLI	**C**ommand-**L**ine **I**nterface

Chapter 1

Introduction

At ISCTE - Instituto Universitário de Lisboa, several services are decentralized and lacks compatibility with mobile devices making it hard to reach using mobile technologies.

The solution proposed in this dissertation involves the creation of a mobile application for ISCTE-IUL students, which provides relevant information and compatibility with most of mobile devices. That information was selected through analysis with future users and the technology available in the institution. This prototype and analysis will help understand the feasibility and the possibility of the application being added to the campus technology.

1.1 Motivation

Several services of ISCTE-IUL are spread across different domains which makes the life harder for students in this institution. They have to *surf* through several websites to find the information they require. Furthermore, another big issue is the lack of compatibility with mobile devices which complicates the life of the students by making them go through a time consuming process that sometimes even the access to the necessary information gets blocked. In several universities in Portugal and worldwide it is possible to find this kind of mobile application helping

their students in their daily basis but unfortunately, ISCTE-IUL still doesn't offer this advantage to its students.

1.2 Relevance

Mobile devices have become trivial to everyone in the daily basis, including college students. The high growth has been forcing several *websites* to adapt to these technologies by using responsive design, which allows the adaptation of their *website* to every mobile devices allowing an easier access to their users. At ISCTE-IUL, only a very small set of services are responsive. This problem leads to many students, which don't have a laptop or don't feel the need of bringing one to the university, having problems accessing the different services and requires them to find an available computer in the campus, which is not always possible.

1.3 Objectives

The goal of this dissertation is the analysis, design and implementation of a prototype which meets the opinion of ISCTE-IUL students and the possibility of those being implemented using the existing services. The prototype integrates several services of ISCTE-IUL in a mobile app accessible to every student in a responsive way for easier visualization and meeting the students expectations.

Chapter 2

State of the art

2.1 Mobile Computation

Nowadays, the use of mobile devices is common in our society and it has been growing at an impressive rate, as figure 2.1 depicts. Using *PORDATA* data[38] we can see that there are over 19 million devices with mobile services in Portugal and most of them are used in a daily basis.

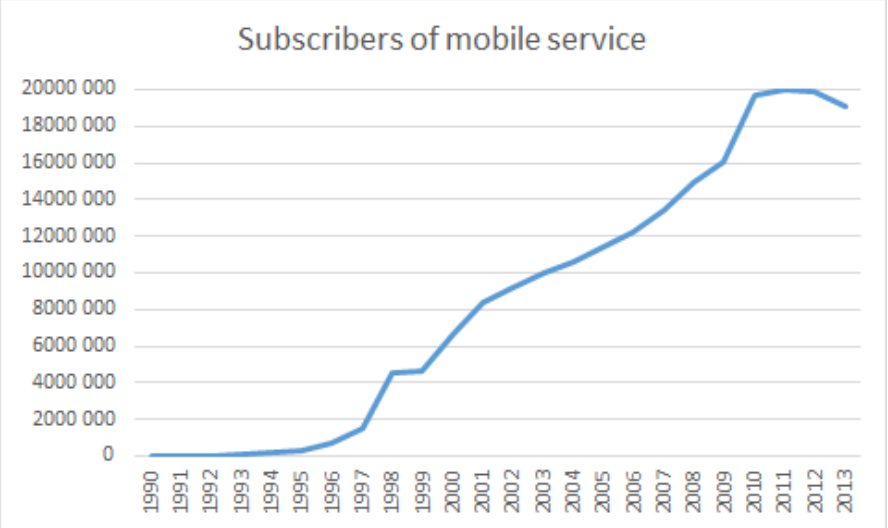

FIGURE 2.1: Chart with numbers of equipments with mobile services, Source: *PORDATA*

Due to this growth, not only in Portugal but over the globe, several new applications emerged for this type of devices. Currently there are over 3 million applications available across four operating systems for mobile devices (see Image 2.2), with most of them for *Android* and *Apple* (data gathered on September 2013 by *Pure Oxygem Labs* [27]).

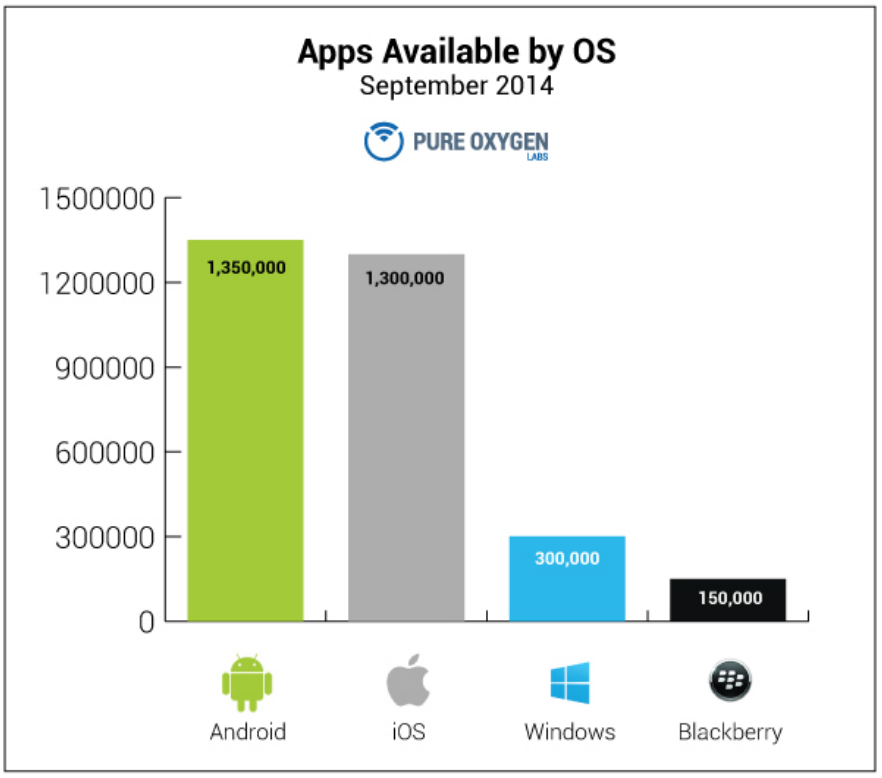

FIGURE 2.2: Chart with number of application in app stores in each OS, Source: *Pure Oxygem Labs*

Everyday, more and more people use their mobile devices for something common as calling or view e-mails, surf on the internet or even shopping. The next figure (see figure 2.3) depicts which online activities people use on their mobile devices, where we conclude that accessing email, texting, searching and social networks have the most attention. This data was obtained through 470 voluntary consumers on *2014 Mobile Behavior Report* from *Salesforce* marketing cloud[29].

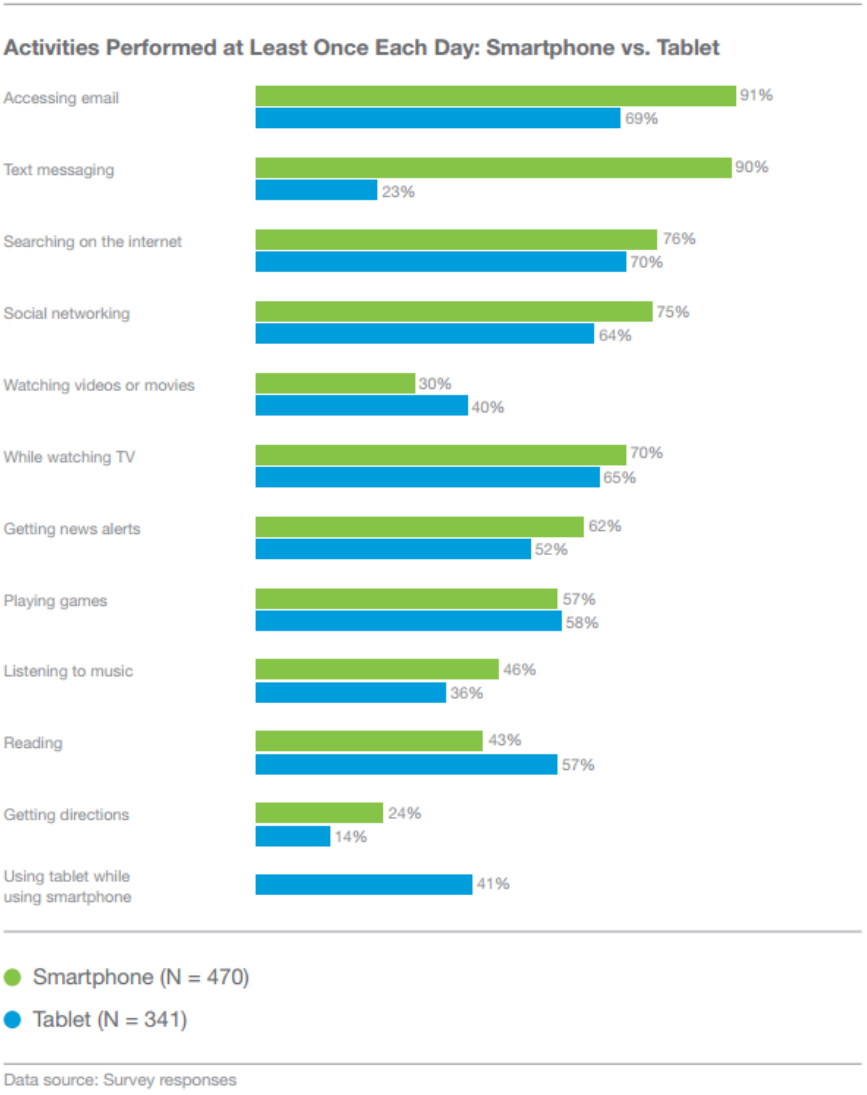

FIGURE 2.3: Chart with percentage of the activity most pratice on mobile devices, Source: *Salesforce*

2.2 Application development for mobile devices

Mobile applications provide more portability and availability, which constitute their main difference to normal desktop-based applications. This type of applications can be easily accessible through application stores, making its distribution a lot easier compared to desktop applications.

Despite its evolution, mobile devices still possess some limitations that affect application development for their platforms as described in *Baymard Institute* article[4].

Among those limitations, we highlight issues like: the compatibility between different versions of the operating system; different screen sizes (size and resolution) across the devices; the computation power and available storage space. In addition to these, a special care is needed in the development process of this type of applications. The lack of context due to the screen size, for example, when answering forms where the user cannot see the whole form content and needs to scroll up and down to view its content. The internet speed can also impact the performance and usability of a application that requires internet access and for example the different interfaces between devices like iPhone and Android, when iPhone uses virtual back button on applications and Android devices use physical buttons.

In mobile devices development there are three types of application that can be developed:

- Native Applications
- Web-based Applications
- Hybrid Applications

2.2.1 Native Applications

Native applications are developed for a specific operating system like *iOS*, *Android*, *Windows Phone* or *BlackBerry*. In the following table (see figure 2.4), obtained in a *ACM Queue* volume[5], it is possible to verify that each operating system possesses a specific programming language, which means that a application that requires to be accessed in multiple operating systems will need more time, work and cost to be available for a larger set of mobile devices users.

Table 1. Required Skill Sets for Nine Mobile OSes

Mobile OS Type	Skill Set Require
Apple iOS	C, Objective C
Google Android	Java (Harmony flavored, Dalvik VM)
RIM BlackBerry	Java (J2ME flavored)
Symbian	C, C++, Python, HTML/CSS/JS
Windows Mobile	.NET
Window 7 Phone	.NET
HP Palm webOS	HTML/CSS/JS
MeeGo	C, C++, HTML/CSS/JS
Samsung bada	C++

FIGURE 2.4: Table with different programming languages for each operating system, Source: ACM Queue[5]

In spite of their disadvantages, native applications bring a superior *user experience* unlike other types of applications, due to its responsiveness and accessibility to native operations like notifications, calls and messages. Moreover, native applications can be easily accessed by their users if available at *App Store* in each OS like *Google Play* and *Apple App Store*.

Some applications like *Facebook*, *LinkedIn*[9] are using this aproach, although initially they have invested in the hybrid approach[41]. This change [6] was based on the fact that it was hard to manage the resources of applications of this scale, for example *Facebook* that possesses several images and videos on their application.

2.2.2 Web-based Applications

Web-based applications have been developed using web technology like *HTML5*, *Javascript* and *CSS* languages. These applications are accessed through *browsers* on mobile devices. Over the years, the potential for this type of application has grown exponentially, much due to the evolution of render engines used in *browsers* (like *Webkit*) and responsive design, which allows the adaptation of the application to every screen.

This type of applications are easy to be accessed, have a lower cost of development in comparison to native applications and the only thing needed to access this type of application is a URL. The interface of these applications is similar to a mobile app. In spite of their advantages, they have some limitations, such as offline (un)availability and lack of access to native functionalities like camera, contacts and offline storage. Besides these limitations, it also lacks the distribution model of App Stores and has to be saved as a bookmark on the browser [20].

2.2.3 Hybrid Applications

Hybrid application development combines the best of native and web-based applications. This type of application has two approaches[42]:

Web Technology which uses *HTML5*, *CSS* and *Javascript* programming languages that runs inside a *WebView* (allowing a application to render a *web* page).[2] To have access to native *APIs* the application uses *Javascript* in the *WebView*. This aproach allows programmers with *web* knowledge to have an easier start on the mobile market. One example of this aproach is the *PhoneGap*[37] framework, which uses *web* technologies like *HTML5*, *CSS* and *Javascript* to create applications for *Android*, *BlackBerry*, *Firefox OS*, *iOS*, *Symbian*, *Ubuntu Touch*, *webOS*, *Windows Phone* and *Windows 8*.

Native Technology applications are developed using one programming language like *C#* and *Javascript* and then are converted to native code for several operating systems.[2] This approach lacks a bit of freedom in their development. One example of a framework of this kind is *Xamarin*[45], that converts *C#* code into *iOS*, *Android*, *Mac* and *Windows* native code.

This type of application provides several advantages like programming in more well-known and largely-used languages while being able to deploy applications to several different platforms, while preserving the responsive design methodology of *web* technologies. This advantage, common to *web-based* applications, can also

take advantage of a native application's features, like access to native *APIs*, larger availability due to distribution model of App Stores and *offline* availability. There are also some limitations since not all native *APIs* can be accessed and the time and cost to make the interface and user experience of the application feel similar to one of a native app can vary. Some well-known examples of such applications are *Instagram*, *Twitter* and *Yelp* which use this aproach[19] through *WebView*, which allows for example *Instagram* to access photos on the device as well as its camera *API*.

2.3 The University Case

Mobile applications in universities is not something new these days and some universities already have their own mobile applications. These applications allow their students to access information that they require in their academic course and information that they can use daily about their universities. This medium is increasingly viable due to the number of mobile devices that their students possess, making it a source not only of useful information for their students but also a great opportunity for universities to explore their potential for sharing events and news to their students.

2.3.1 Examples of universities mobile applications

Some universities already have mobile aplications available to their students, such as the University of Oxford[32], Harvard University[44], Massachusetts Institute of Technology[33], University of Illinois Springfield[31], University of Massachusetts Boston[30], Universidade de Aveiro[7] and Universidade Católica[39]. These universities chose to use the hybrid approach, allowing it to be accessed by a large number of their students, with less development cost unlike native applications. There are some universities like Lusófona[28] and Instituto Superior Técnico[8] that chose to use native applications but for now they only offer an application for *Android* devices.

2.3.2 Features grid and comparison

After performing an analysis on national and international mobile applications (see Table 2.1), it is possible to see that the applications that have been developed using the hybrid model (like Oxford, Harvard, UIS, UA, UCP) have the same or more features than Lusófona and IST native applications. No single application has implemented every feature on the list, although some features like library, canteen, community (chat, forum) and student schedule were the most used in this universities.

Features \ Universities	Oxford	Harvard	UIS	UA	UCP	Lusófona	IST
Library	x	x	x		x		
Canteen		x	x	x			x
Course Catalog	x	x					
Community (Chat, Forum)	x		x	x	x	x	
Contacts	x	x	x				
Sports Teams		x					
Parking Lot	x			x	x		
Events	x	x				x	
Students (Tuitions, Grades)				x	x	x	
Student Schedule			x	x	x	x	
Transports Schedule	x						
Emergency Information	x		x		x		
Map	x	x	x				
Weather	x						
Multimedia		x					
News	x	x		x			
Academic Services				x		x	
Shuttle		x					x

Table 2.1: Features of mobile applications in some national and international universities

2.4 Conclusion

With the growth of mobile devices and their use, several applications were created to help the consumers in their daily activities. Several universities took this opportunity to integrate their services in this type of applications, making it available to their students.

Analyzing the mobile development, three types of applications have been found. Native applications, which run inside the mobile devices, allowing a total access to the device but they require more time and cost in their development since each operating system possesses its own programming language. *Web-based* applications, that although being easy to develop using *web* technologies like *HTML5*, *CSS* and *Javascript* lack availability and access to native *APIs*. And at last, *hybrid* applications, which can be developed by using *web* technologies or one single programming language and then can be converted to several *OSs*, allowing a faster development and access to some native *APIs*.

On the university context, these applications offered several features for their students, creating them an easier access to services of their university. Not all the applications use the same technology, but the hybrid approach clearly surpasses the native approach. In spite of their differences, these applications brought a similar context to the user.

Chapter 3

Application Concept

Based upon analysis in chapter 2, it was decided to use a hybrid approach in the implementation of this application.

This chapter details the current situation at ISCTE-IUL regarding information systems, the students opinions on this matter and performs an analysis of several frameworks available for hybrid applications. Furthermore, based on the requirements that were defined, on the opinions of the students and on the frameworks that were analysed an application conceptualization was developed and can also be found later in this chapter.

3.1 Current Situation

Presently, at ISCTE-IUL, some of the digital services available to the students are dispersed through different systems. This makes it difficult for the students to find the information they need without using multiple *websites*, which takes more time than it should.

Moreover, these websites do not have a responsive interface, which means that their use is limited in mobile devices with small screens. This problem forces the

students to access their university services with laptops or institutionally-provided desktops to access the information.

This type of applications is available throughout Portugal in some high education institutions but ISCTE-IUL hasn't developed a mobile application for their students yet.

3.2 Students Opinion

In October 2014, a questionnaire was made to students at ISCTE-IUL and 246 of them answered the questions. The majority of the students were male (64%) with ages between 21 and 25 years old. (see Image 3.1 and 3.2)

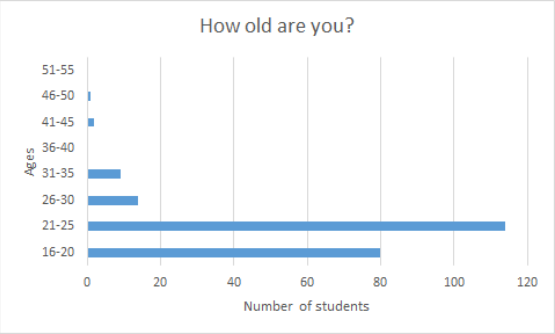

FIGURE 3.1: Chart with ages of the students

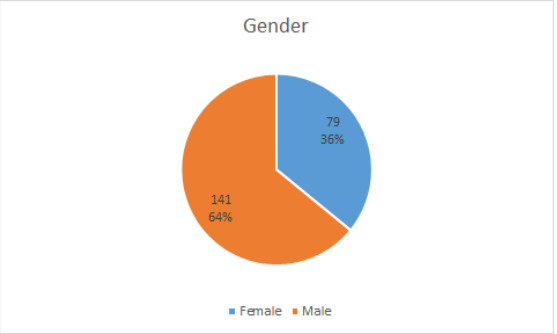

FIGURE 3.2: Chart with genre of the students

The majority of the students were from the engineering or computer-related degrees but there is a large distribution of answering students across the other institution's degrees (see Image 3.3).

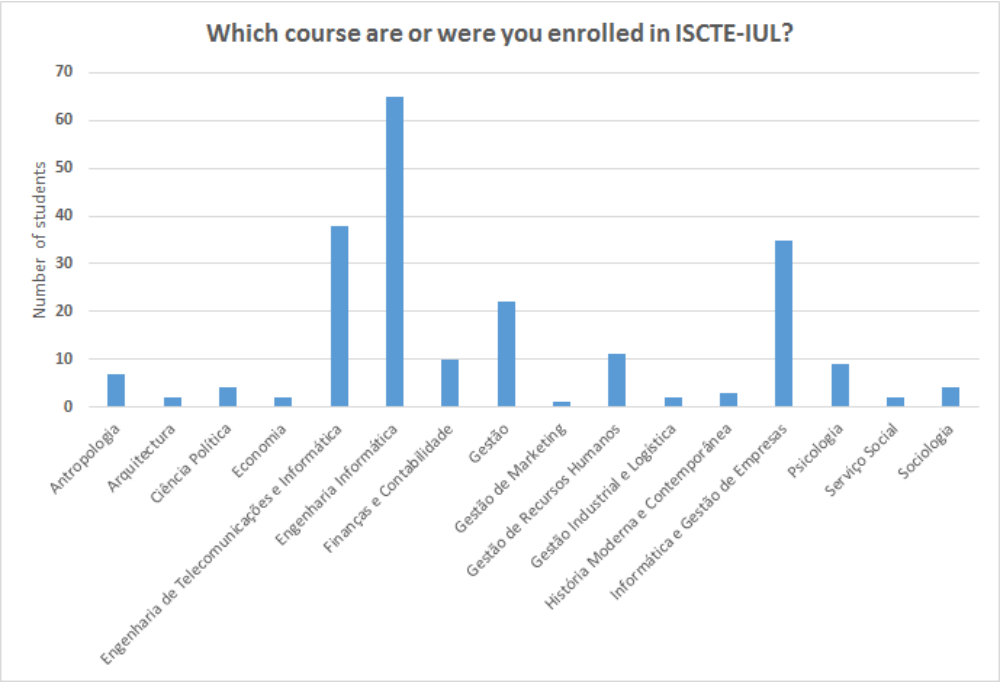

FIGURE 3.3: Chart with the students enrolled degrees

From all questioned students, around 229 (93%) have a *smartphone* (see Image 3.4), but 72% access the university services with laptop instead of their smartphones (see Image 3.5).

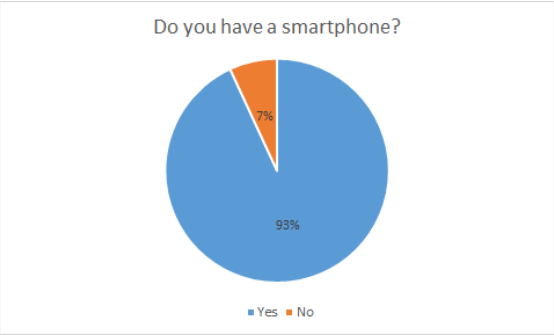

FIGURE 3.4: Chart with students that have a smartphone

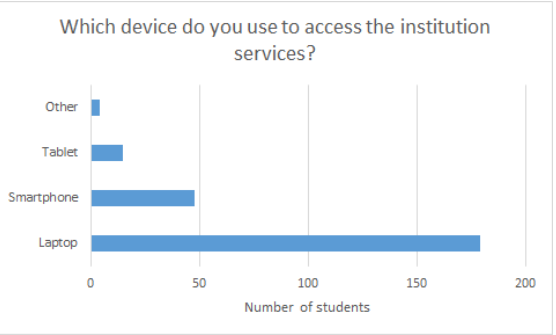

FIGURE 3.5: Chart with devices used to access university resources

Most of the questioned students have at least one *smartphone* with *Android* (68%) followed up by *iOS* (20%) (see Image 3.6). About a mobile application, 231 students (94%) said that they would consider a mobile application of ISCTE-IUL helpful for their academic degree. (see Image 3.7).

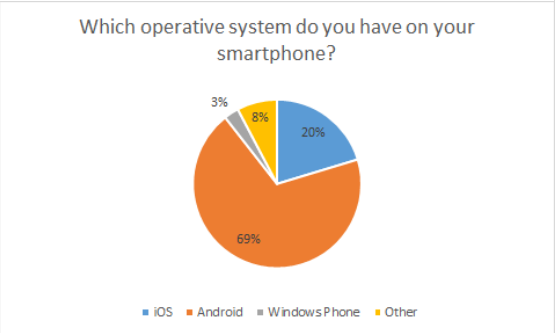

FIGURE 3.6: Chart with the distribution on operative systems

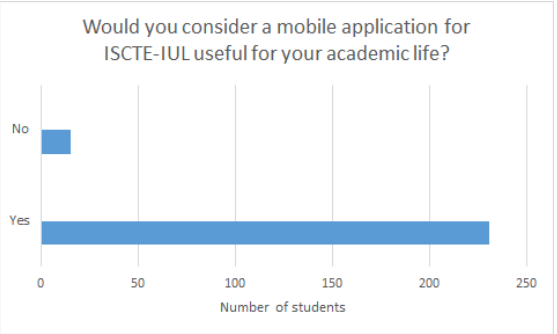

FIGURE 3.7: Chart about the usefulness of a mobile application on ISCTE-IUL

Questioned about the features they would like to see in the mobile application, with a scale from 1 to 5, the most voted were the ticket service of the secretariat (4.39), access to the student's private information (4.31) and canteen (4.08). From all the features, the least desired was the Online Shop (2.98) (see Image 3.8).

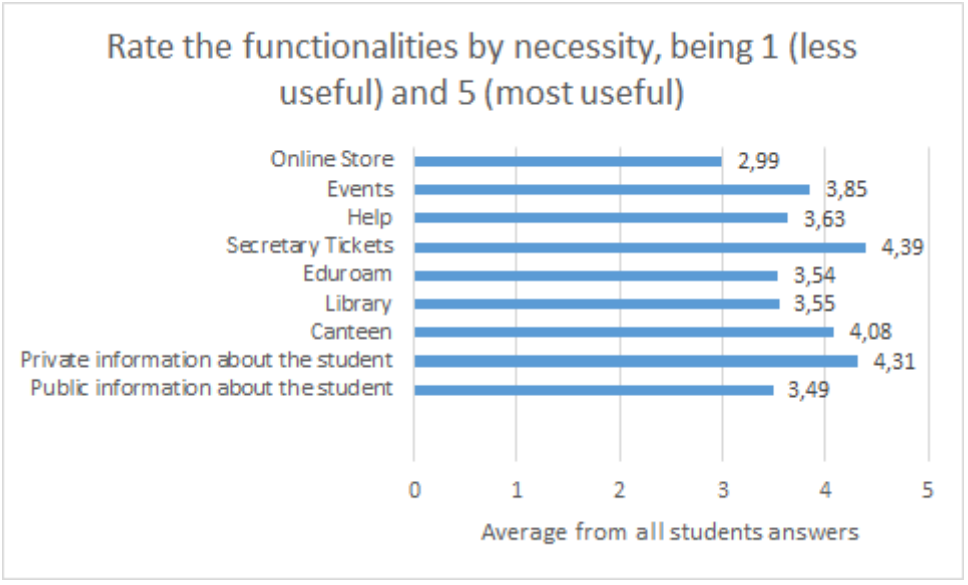

FIGURE 3.8: Chart with the preferred features by the ISCTE-IUL students

3.3 Application Requirements

3.3.1 Functional Requirements

After gathering the opinion of the students and analyzing the available services at ISCTE-IUL, it was possible to define the requirements of the application that are useful to the students and also possible to develop.

Canteen students can see the menu of the next 5 days.

Students Informations public information about the students where it is possible to see their chairs and teachers.

Teachers Informations teacher information regarding email address, locker, office, all chairs lectured and curriculum vitae.

Secretariat Tickets students can see tickets information, called numbers and desks.

Map possibility to select a classroom or points of interest at iscte-iul and show them in google maps API.

Transports see next strikes and transports near ISCTE-IUL.

Furthermore, some information used on people daily basis were added.

Weather possibility to see the weather at the university's location.

News about the university and national news.

3.4 Framework

In chapter 2, it was possible to determine that a hybrid application fits best in this project, making it easy to develop without the need to use multiple languages for

the different OSs, reducing its cost and time. However, there are several different ways and frameworks to build a mobile application. This section analyzes some of these different alternatives and presents the chosen approach.

3.4.1 Xamarin

Xamarin is a cross-platform framework that allows you to make apps for *iOS*, *Android*, *Windows* and *Mac*. It uses *C#* as their language and converts the code for each specific platform, letting it use hardware acceleration and UI conventions. It is pointed that 60-100% of the code can be reused[45]. Although it supports *MVC* patterns and recent wearables like Android Wear, a subscription is required to use the framework and deploy the developed applications.

Some examples of companies that uses this framework are *Microsoft*, *Foursquare* and *Bosch*.[46]

3.4.2 Titanium

Titanium is an open-source product of *Appcelerator*, which uses *Javascript* as their language and does not allow *HTML5* for things like animations. It uses Native APIs and adapts to each OS UI conventions.

Titanium allows to build apps for *Android*, *BlackBerry* and *iPhone*. Although it is possible to reutilize about 100% of the code for the different platforms, this is a rare situation and it is pointed on the *Appcelerator* website in the majority of the case the code reutilization is only about 60-90%, with special attention being needed for each platform.[36]

Applications have a user experience similar to native and it's free to try but unfortunately it is required to pay for deployment and not recommended for large scale applications. A few examples of companies who use this framework and tools are *Cisco*, *VMWARE* and *Mistubishi Electric*.

3.4.3 PhoneGap

PhoneGap is a cross-platform which uses web technology to develop a mobile application. This application is rendered through a Web View on the mobile device. This framework is open-source and uses *HTML5*, *Javascript* and *CSS3* and allows to build apps for *iOS*, *Android*, *Windows Phone 8*, *Blackberry*, *Firefox OS* and *Ubuntu*.

The code is highly reusable for the different platforms and their *APIs* are simple to use. Unfortunately, this type of applications tend to have worse performance and their UI will depend on the Web View rendering system.[37]

3.4.4 Comparison Grid

These three frameworks are the biggest in today's market, and provide the best option for hybrid application development. Each framework has its own features, which are described in the table below made by *Cygnet Infotech*[37] (see Table 3.1).

Features \ Framework	PhoneGap	Titanium	Xamarim
Platform Support	iOS, Android, Windows Phone 7 & 8, BlackBerry	Android, iOS & Blackberry	iOS, Android & Windows
Language	HTML5, CSS3, JavaScript	JavaScript	C#
Open-source	Yes	Yes	No
UI	Web UI	Native	Native
Access to device *API*	Limited	Full	Full
Web Standart Support	Yes	No	No
DOM Support	Yes	No	Yes
Native Performance	No	Yes	Yes
Used By	IBM, Sony, Mozilla, Intel	Cisco, VMWARE, Safeguard Proprities, Mitsubishi Electric	GitHub, Microsoft, Foursquare, Expensify, Dow Jones

TABLE 3.1: Features for the different Framework discussed, Source: Cygnet Infotech

PhoneGap allows the developers to have a larger potential user base by allowing the deploy to be made in several different OSs, whereas *Titanium* and *Xamarin* only allow Android, iOS and Blackberry. Out of the three, only *PhoneGap* uses HTML5 and CSS3 which allows to make animations using its Web View rendering system. *Xamarim* is not open-source, but both *Titanium* and *Xamarim* require a payment fee to develop and/or deploy the application.

All of them have access to the mobile device *API*, but *PhoneGap* has a limited access due to the application running in a *WebView*, making it also the only one that doesn't have a native performance and is limited by its rendering system.

3.4.5 Ionic

After *Adobe* acquired *PhoneGap*, the open source core system was donated to *Apache Software Foundation* with the name *Cordova*. This core system is used by *Ionic* team, *Drifty*, and is the core for the *Ionic Framework*.

Ionic provides a *HTML5 SDK* that optimizes the native feeling of their applications and it's focused on the front end of the application. It utilizes *AngularJS* and besides allowing *CSS* customizations, it also offers UI components with native app feel. Although it is in its early steps, the framework has been proved powerful and its stable version has been released in May, 2015. Since it has a MIT license, any developer can use, build, modify and publish free of any cost. *Drifty* has been providing additional services to maintain the framework, like *Analytics*, *Push* and *Deploy*.

3.4.6 Decision

In spite of having great features and characteristics, *Xamarin* and *Titanium* have an associated cost that needed to be cut down. Furthermore, the knowledge acquired from other projects on *Web Techologies* made *Cordova* the framework to go for this application, since it is possible to develop all the features required in this framework. As a result of further investigation, it was determined that *Ionic* features enhance *Cordova* making it the ideal framework to use in this project.

3.5 Facing Requirements

One of the main issues with this type of applications, is the need to be always online due to frequent change of information.

This dependency makes it critical to have a high server uptime and more control over *APIs*. To make this more stable, it was required to make the *APIs* information stored in a *RESTFUL API* created for the application, which can be used for other applications inside the university. This way it is possible to increase the availability of the application even if there are any problems with the server providing the information. This information is stored in a database inside the institution making it faster to access due to its location to provide a lower response time. This decision impacts the user experience by providing a faster response to the user.

The amount of different devices available in the market is also an issue, since there are several combinations of specifications and different OS. This issue reflects in compatibility and also performance.

Unfortunately some of the most required features of ISCTE-IUL students based on their opinion earlier in this chapter could not be developed in this application since there is no *API* to allow access to private information like schedules, exams and tuitions.

3.6 Conceptualization

This section contains the conceptualization of the application using the chosen framework and it covers the *APIs* used as well the database created for this application.

3.6.1 ISCTE-IUL APIs

For the prototype of this project, ISCTE-IUL allowed the use of one of their development servers to access some of the students public information. This information is passed through a Web Service that connects the Fenix platform and Blackboard. This information was crucial to the creation of this project.

With this API, it was possible to gather information about:

Students Personal information about the email, username. Crossing data allows to see the student degree, and curricula that the student is currently enrolled in.

Teachers Personal information similar to students. Crossing data allows to see curricula which the teacher lectures.

Curricula Information about the name of the course in English and Portuguese as well the code to identify the course. Crossing data allows to see which degree the curricula belongs to, which students are enrolled and who are the professors lecturing the curricula.

Courses Information about the name of the degree in Portuguese and English and the identifying code.

To access this web service a script has been made to load the information on a scheduled job. The script was developed using *PHP* to request the *API* the information using *SOAP* (Simple Object Access Protocol) envelope with a *WSDL* (Web Service Definition Language) description.

Unfortunately the web service does not have the functionality of only providing the updated information, which requires the script to clear the database and insert the entire data set again.

Another *API* used is *Ciencia-IUL*, which provides more detailed informations about the teachers of the institution, where students can see their office, postbox,

email, Curriculum Vitae and the areas in which they do research or are qualified to supervise grad students. The documentation of this *API* can be accessed through *Ciencia-IUL*[21] website.

Finally, the *MyTicket*[22] *API* was also provided by ISCTE-IUL allowing the application to view all the queue lines for the services in the institution.

3.6.2 Third-Party APIs

Besides the *APIs* inside ISCTE-IUL, some third-party *APIs* have been used to provide relevant information to the user about public transportation strikes and weather.

The strikes *APIs* is provided by **Ha Greve**[23]. The creators of this website gather information around the internet from official companies websites and news stations to inform their users of the upcoming strikes. Their *API* is currently in its second version allowing access to companies, all strikes or only the upcoming strikes. The *API* response is in *JSON*, allowing an easy interpretation by Javascript converting the results in arrays.

The other *API* is provided by **OpenWeatherMap**[35]. **OpenWeatherMap** provides a *API* with weather data using broadcast services, airport weather station and other official weather stations. This *API* can be used for free and allows to request information about the current and forecast of the weather for a single or multiple location. In the application developed in this project the location is set to Lisbon, where ISCTE-IUL is located. Similarly to the strike *API*, the response is in *JSON*.

3.6.3 Use Case Diagram

After reviewing the requirements, the available *APIs* and their respectively information it was possible to determine two actors in the application:

- Student
- Administrator

The students in the application will be allowed to view information regarding themselves, their teachers and public information provided by the application. Administrator is an actor that it is needed to update some of the information that is not possible to be obtained through *APIs*. This moderator however can be removed if the ISCTE-IUL infrastructure creates an *API* for this purpose (see Image 3.2).

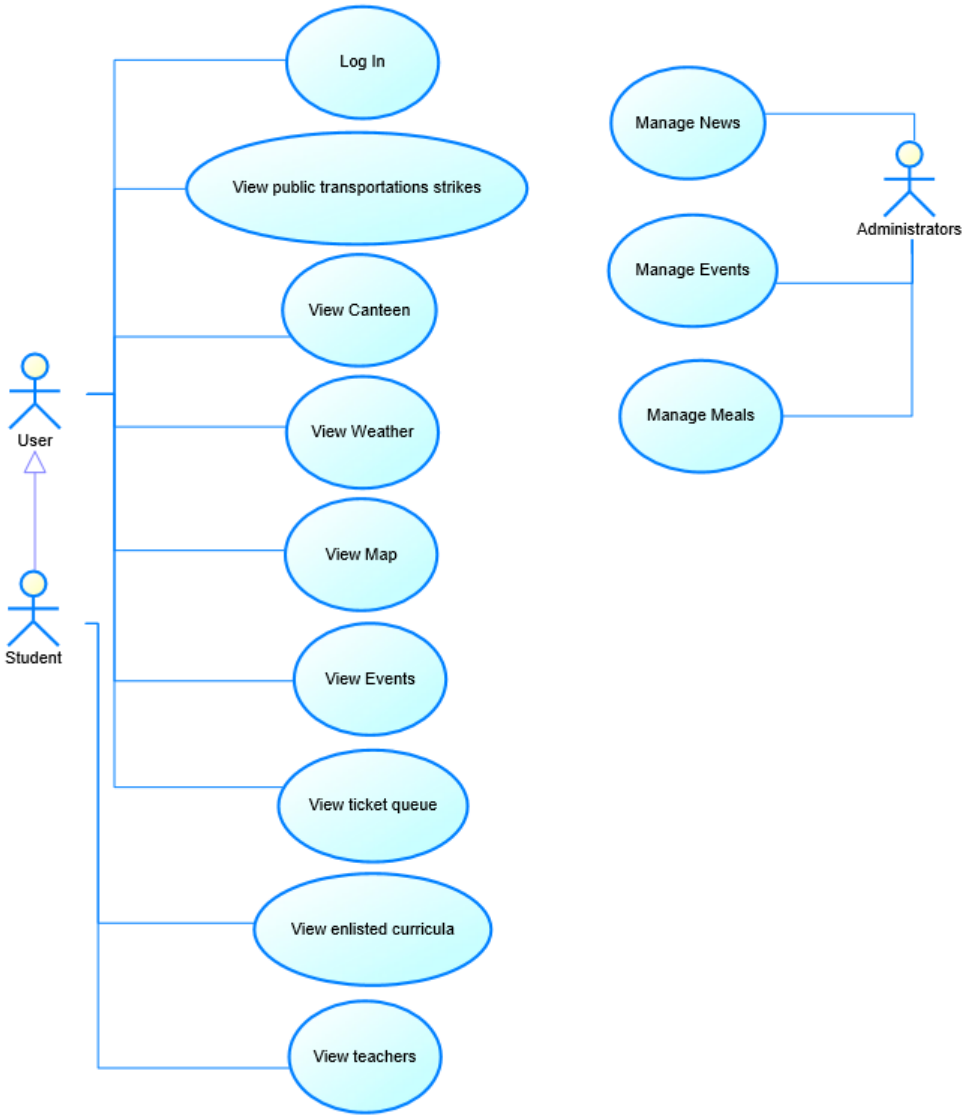

FIGURE 3.9: Use Case Diagram of the application

In the table below (see Table 3.2) it is possible to find a more detailed description about each use case.

Name	Description
Log In	The user authenticates to the server and if successful he becomes the actor student.
View public transportation strikes	User can view if there are any strikes in the upcoming days
View Canteen	Allows the user to view all the meals for the upcoming days
View Weather	Allows the user to view the weather for Lisbon in the following days
View Map	Allows the user to view all the POI around ISCTE-IUL and their location
View Events	Allows the user to see all the events informations for the following days
View ticket queue	Allows the user to check the queue line for the institution services
View enrolled curricula	Allows the student to see all the disciplines that he is enrolled
View teachers	Allows the student to see all the teachers that lecture the disciplines that he is enrolled
Manage News	Allows the administrator to create, edit and remove news.
Manage Events	Allows the administrator to create, edit and remove events.
Manage Meals	Allows the administrator to create, edit and remove meals.

Table 3.2: Detailed description of the use cases

3.6.4 UML Diagram

As described earlier in this chapter it was required a database to maintain the information so that the application only sends requests to a single service, reducing its response time as well as granting application availability if one of the services inside ISCTE-IUL or third-party *APIs* goes to maintenance or is even down.

This database needed to contain all the information related to ISCTE-IUL, as well as some services information. This being said, all the information related with the students, curricula, teachers, canteen, map, news, events and even public transportation strikes has been stored in the database. Since weather is a real-time information, it does still pass through the *API*, but it isn't stored in the database (see Image 3.10).

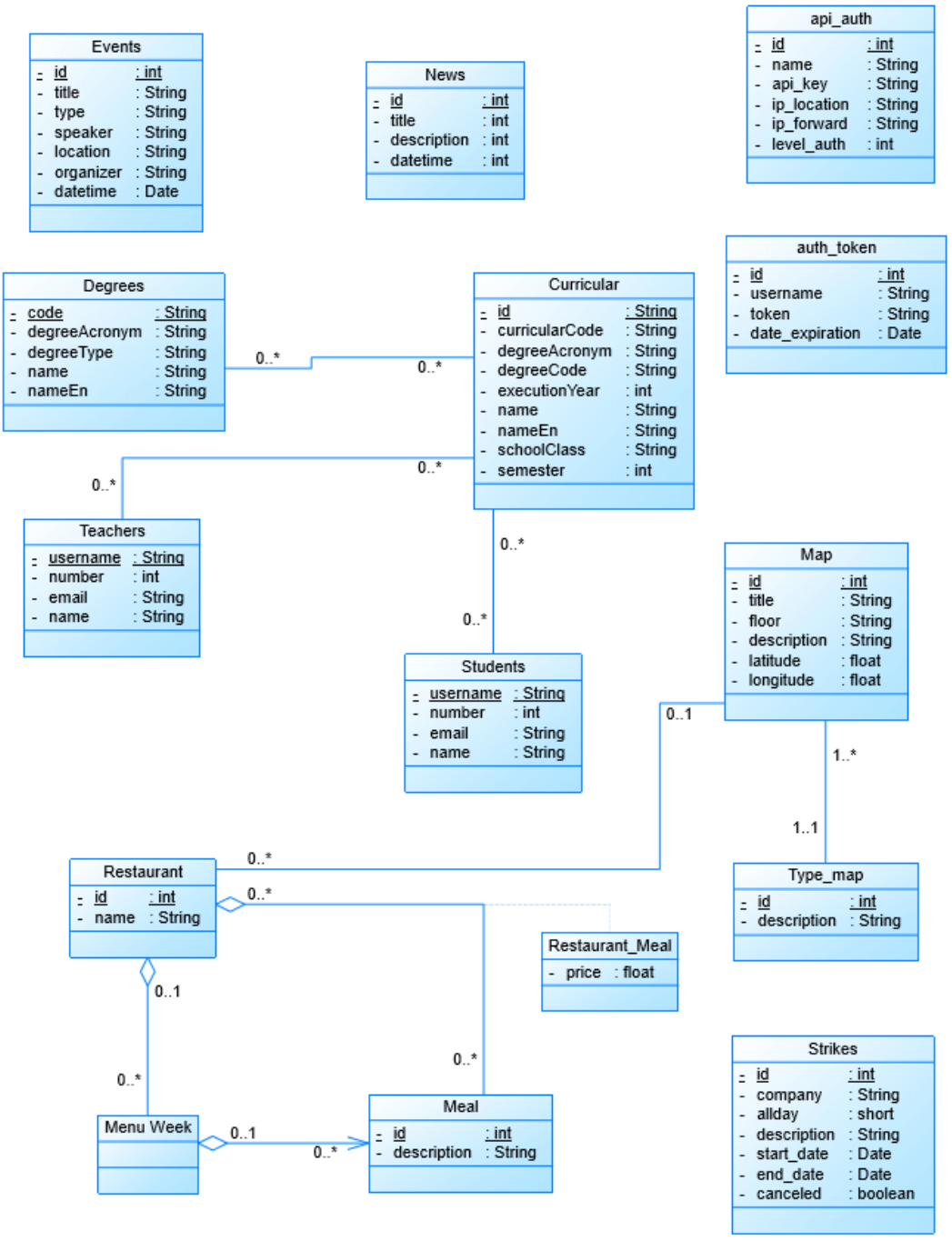

FIGURE 3.10: Class diagram of the application

A more detailed explanation of the choice in this diagram is provided in the following subsections below.

3.6.4.1 Curricula

Initially, the database was planed to save all the information about past years, similar to a case study found in **Pedro Nogueira Ramos** book entitled **"Desenhar Bases de Dados com UML"**[40] (see Image 3.12). After some thought it has been defined that the *API* shouldn't contain the past information and only relevant information about the current year, so an adaptation has been made (see Image 3.12).

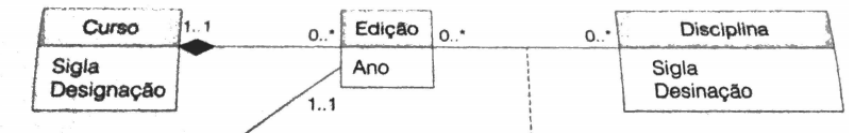

FIGURE 3.11: Original case study Class Diagram

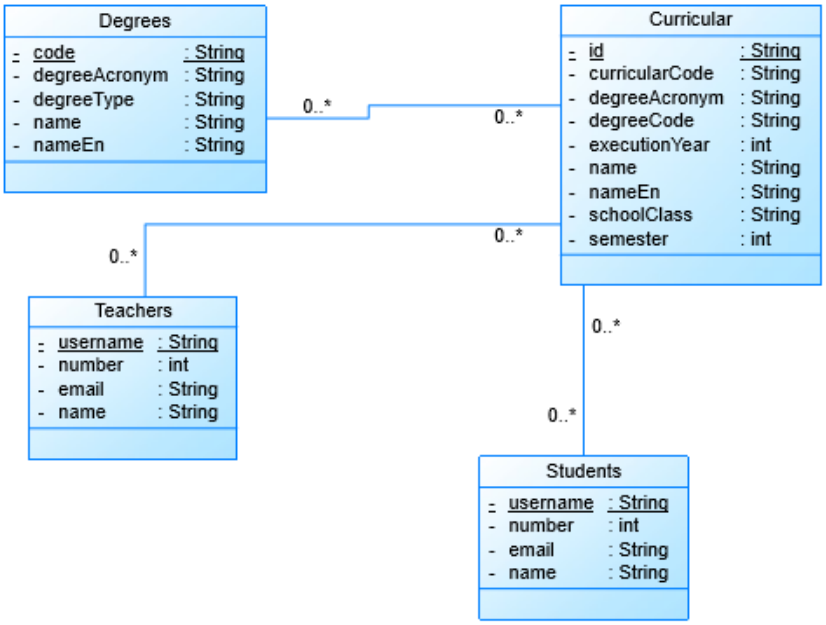

FIGURE 3.12: Curricula Class Diagram adaptation for the application

In the adaptation of the diagram seen in the book, the **Edition** table has been removed, as well as its composition since it isn't required for the *API* to work and would overflow the database.

3.6.4.2 Canteen

In ISCTE-IUL there are several places to eat and even restaurants. This information can be made available in the application benefiting the students and also giving visibility to the restaurants (see Image 3.13).

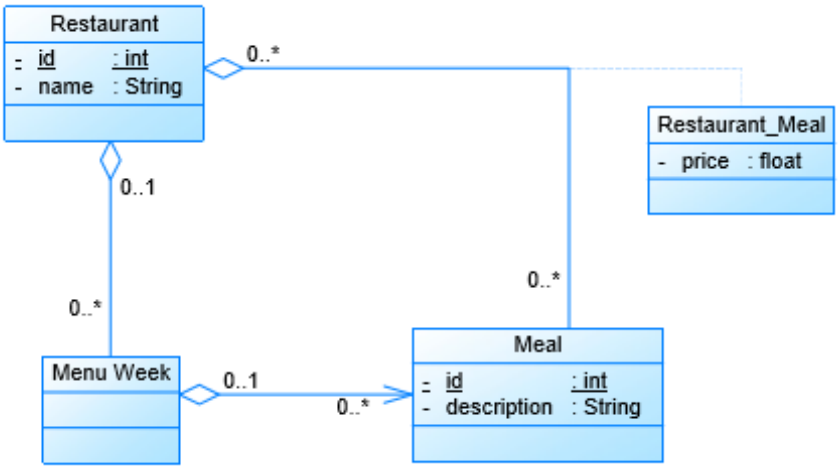

FIGURE 3.13: Canteen Class Diagram of the application

In this model it is possible for a restaurant to have dishes of the day and also meals available all week long, making it possible to see their prices.

3.6.4.3 Others

The rest of the database contains miscellaneous information which are separated from the other tables (see Image 3.14).

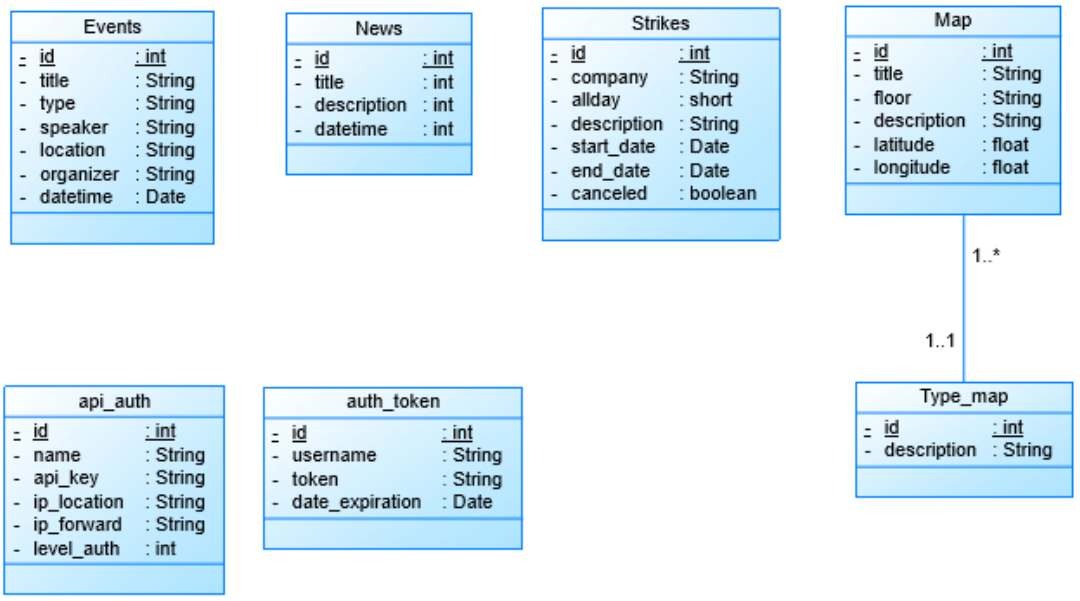

FIGURE 3.14: Miscellaneous Class Diagram of the application

Events Events contain all the events coming at ISCTE-IUL.

News News contain the news about the institution

Strikes Strikes table contains the public transportation strikes that are schedule for the next days.

Map & Type_map Table contains the latitude and longitude of several POI around ISCTE-IUL.

api_auth & auth_token Security tables that will authenticate and verify both the device and the user.

3.6.5 Restful API

Earlier in this chapter it was possible to detect all the *APIs* available and the database needed to provide the information requested by the application. To improve the stability and availability a *RESTful API* was developed using **Slim Framework** to request information from the database.

Slim is a micro PHP framework that simplifies the creation of powerful web applications and *APIs*[25]. Similar to Ionic, it also possesses a *MIT* license, allowing to use, build, modify and publish free to any cost.

Due to security reasons, all requests and responses for the third-party informations like weather and other future *APIs* used are passed through the *RESTful API*.

3.6.5.1 Security

Some information is specific for each student and should only be accessed by that student. To ensure that the information only goes for that specific student, every student has to login to access certain parts of the application. They login using *LDAP* (Lightweight Directory Access Protocol) system available inside the institution which authenticates the user. The *API* after verifying the login sends a token to the application. This token has an expiration date and after expiring the user needs to authenticate again.

Besides the user authentication, another authentication is required to access the *API*. Each application or system requires a token or a registered IP to have access to the *API*. This verification prevents unwanted access or overload to the *API*.

Chapter 4

Application Development

In this chapter it is possible to read all the described work behind the development of the application, since the initial learning of the chosen languages to the final production of the prototype.

4.1 Database

The corresponding database of the Class Diagrams defined in the concept phase (available in the previous chapter) was implemented in a *MySQL* server inside a Raspberry PI 2 for the duration of this project. After the implementation it was required to populate the database with information.

4.1.1 Populating Database

Thanks to ISCTE-IUL help, the access to an *API* that connects E-Learning platform and Fenix platform was provided allowing a more authentic prototype using real students information. The specific *API* uses *SOAP* and a *WSDL* describer.

To populate the database it was required a *PHP* script, which gathers the information of the ISCTE *API* by the following steps:

1. Retrieve and insert all teachers and their informations and insert in the database.
2. Retrieve and insert all degrees.
3. Retrieve and insert all students by degrees.
4. Retrieve and insert all curricula by degree.
5. Insert link (enrollments) between student and curricula.
6. Insert link (professorship) between teacher and curricula.

Furthermore, when each teacher is retrieved from the ISCTE-IUL *API*, another *API* is called (Ciencia-IUL) which provides even more information about the teachers, which for the application were selected the Postbox number, Office and Curriculum Vitae.

4.1.2 Limitations

Due to the ISCTE-IUL *API* providing all the information every request and not only the updated information, it was required to clear the database and populate again each call to guarantee that all the information is up-to-date.

Another alternative was to make a verification in each row (each teacher, student, degree and curricular) and check against information already in the database. This process would take even larger time to process to an already slow process due to the number of records in the database, so for that reason was discarded as a viable alternative.

4.1.3 Generating PHP classes

Similar to how it is done in JAVA, a set of classes can be generated to connect to a *SOAP/WSDL API*. This way allows for an easier interaction between *PHP* and

SOAP as complex types used in *SOAP/WSDL* can make this task more difficult. For this purpose, a code generator was used, called *EasyWSDL2PHP*[43]. With a few modifications for the front-end interface, it was possible to generate a file named *ElearningSystemService.php* with all the methods available, its parameters and classes both for search and response.

Below are the methods and classes to retrieve teachers using ISCTE-IUL API.

```
class getTeachers{
        var $arg0;//teacherSearchDTO
}
class teacherSearchDTO{
        var $executionYear;//int
        var $username;//string
}
class getTeachersResponse{
        var $return;//teacherDTO
}
class teacherDTO{
        var $email;//string
        var $name;//string
        var $number;//int
        var $photo;//base64Binary
        var $username;//string
}

function getTeachers($getTeachers)
{

        $getTeachersResponse = $this->soapClient->getTeachers($getTeachers);
        return $getTeachersResponse;

}
```

The first class **getTeachers** is the object required (parameter) for the **getTeachers** function. This class requires a **teacherSearchDTO** which allows to be filtered by **username**. Although in this generated code, there is no information as to which variables are required or not, the **executionYear** is mandatory and without it the function will not be called. After being called, **getTeachersResponse** is returned. This class only cointains a variable called return which is an

array of **teacherDTO**. This Data transfer Object possesses all the information about the teacher.

4.1.4 Script

As described before, this script required to clear all the tables which information is provided by ISCTE-IUL *API* to guarantee its updated information. To do this, and since all the information for this database will be cleared, the foreign key verification is temporarily disabled.

```
SET FOREIGN_KEY_CHECKS = 0;
TRUNCATE curricular;
TRUNCATE degrees;
TRUNCATE degrees_students;
TRUNCATE students;
TRUNCATE teachers;
TRUNCATE enrollments;
TRUNCATE professorship;
SET FOREIGN_KEY_CHECKS = 1;
```

It was created a **DatabaseHelper** class that contains all the methods to populate the database. This class uses the **ElearningSystemService.php** generated class to gather the information. Below is an example of how the class obtains and inserts the information of the teachers in the database.

```
class databaseHelper{
private $conn = null;
private $eLearning = null;

function __construct() {
$host = "localhost"; // change this as required
$username = "root"; // change this as required
$password = "PASSWORD"; // change this as required
$db = "thesis"; // your DB


// Create connection
$this->conn = new mysqli($host, $username, $password, $db);
// Check connection
if ($this->conn->connect_error) {
        die("Connection failed: " . $this->conn->connect_error);
}
```

```
$this->eLearning = new ElearningSystemService();

}
```

The constructor makes the connection to the database, in this case both are localized in the Raspberry PI 2 and after initializes a new variable with **ElearningSystemService** class.

```
function insertTeachers(){

$searchTeachers = new teacherSearchDTO();
$searchTeachers->executionYear = 2014;
$paramTeachers = new getTeachers();
$paramTeachers->arg0 = $searchTeachers;
$teachers = $this->eLearning->getTeachers($paramTeachers);
$listTeachersDTO = $teachers->return;
foreach ($listTeachersDTO as $helper)
{
$helper->email = mb_convert_encoding($helper->email, 'iso-8859-1', 'utf-8');
$helper->name = mb_convert_encoding($helper->name, 'iso-8859-1', 'utf-8');
$helper->username = mb_convert_encoding($helper->username, 'iso-8859-1', 'utf-8');

$helper->email = $this->conn->real_escape_string($helper->email);
$helper->name = $this->conn->real_escape_string($helper->name);
$helper->username = $this->conn->real_escape_string($helper->username);


if(!empty($helper->photo)){
$imagepath='/var/www/testing/api/images/'.$helper->username.'.png';
file_put_contents($imagepath, $helper->photo);
}


if($helper->number > 0){
$ciencia = file_get_contents('https://ciencia.iscte-iul.pt/api/author/'.$helper->number);
$obj = json_decode($ciencia);

if(isset($obj->error_msg)){
$SQLstring = "INSERT INTO teachers (number, email,name,username)
VALUES ('$helper->number', '$helper->email', '$helper->name', '$helper->username')";
}
else{

$office = $this->conn->real_escape_string($obj->author_info->contacts->office);
$postbox = $this->conn->real_escape_string($obj->author_info->contacts->post_box);
$cv = mb_convert_encoding($obj->author_info->cv_research->cv_summary, 'iso-8859-1',
```

```
 'utf-8');
$cv = $this->conn->real_escape_string($cv);
$link = $obj->ciencia_iul_url;
$SQLstring = "INSERT INTO teachers (number, email,name,username,office,post_box,cv_summary,
link_cienciaviva) VALUES ('$helper->number', '$helper->email', '$helper->name',
'$helper->username','$office','$postbox','$cv','$link')";

}
}
else{
$SQLstring = "INSERT INTO teachers (number, email,name,username)
VALUES ('$helper->number', '$helper->email', '$helper->name', '$helper->username')";

}

if ($this->conn->query($SQLstring) === TRUE) {
//      echo "New record created successfully";
} else {
echo "Error: " . $SQLstring . "<br>" . $this->conn->error."<br>";
}

}
//      echo "<br><br><br><br>Completed teachers!";
}
}
```

When called, the **insertTeachers** function makes a request to the API following the steps described above, which returns an array of **teacherDTO**. With each iteration of the array, it encodes the **charset** to **utf-8** to avoid special characters problems and then if the teacher has a picture, place it in a file. Another alternative for this was to place it inside a blob in the database. Using the **Ciencia-IUL** *API*, it collects even more information of the specific teacher and inserts in the database.

4.2 Restful API

After the creation of the database and script to populate it, the next step was to provide a way of communication between the database and application. In the concept phase it was decided that an *API* would be required for this project due

to the constant change of information and a system authentication to allow the user access to his information.

For this *API*, it was also defined that *Slim Framework* would be used to accelerate its development.

4.2.1 Slim Framework

Slim is an *MIT*-licensed framework which code can be found in their website[25]. The framework is on its second version but the third version is already on the beta phase featuring **Dependency injection**, **PSR-7 support** and more[24].

The programming language of this framework is *PHP*.

4.2.2 PHP

Although past experience with *PHP* language some of its features like **PHP Data Objects** (PDO) statements haven't been used before. The same happened with *Slim* since it was the first experience with the framework.

4.2.3 Where to Start

After downloading the framework it is possible to find a Slim folder which contains all the core files to the *Slim* framework and an **index.php** file. All the code for the *Restful API* can be coded in the **index.php** file.

The *Slim* framework is used as a class in the **index.php** file. To start off creating the *API* it is required the following code:

```
require 'Slim/Slim.php';

\Slim\Slim::registerAutoloader();

$app = new \Slim\Slim();
```

As described in the **index.php** file it is need to include the **Slim/Slim.php** to load the *Slim* framework. After that, simply initialize an object with *Slim* class. Now it only requires creating routes for the application depending on the HTTP request method which are *GET, POST, PUT, PATCH, DELETE.*

Below is a simple example of how to redirect these requests to functions and return a *"Hello World"* with the name provided by the request using *GET* and *POST*.

```
$app->get('/helloworld/:name', 'getHelloWorld');
$app->post('/helloworld', 'postHelloWorld');

function postHelloWorld() {
        global $app;
        $req = $app->request();
        $name= $req->params('name');
        echo "Hello, ".$name;
}

function getHelloWorld($name) {
        echo "Hello, ".$name;
}
```

As seen in the code, two routes were created which can be pointed to one or multiple functions. One for a *GET* request, essential trigger if a user visits **HTTP://URL/helloworld** and a *POST* request. In the first line, where the *GET* route is created it is possible to see a **:name** at the end of the first parameter and can be provided by placing in the end of the URL /**John** so the URL would be **HTTP://URL/helloworld/John** This tells us that a variable is required and that it will be passed to the **getHelloWorld**. In the *POST* request it is necessary to get all the request of the *API* and get the required parameters. To finish, after all the routes are defined it is essential to run the *API*.

```
$app->run();
```

4.2.4 Database Connection

The *API* need to be connected to the database to provide the information required to the application. To be able to do this a database connection is required in each request.

Each method that requires an operation to the database calls the **getConnection** method listed below.

```
try {
        $db_username = "USER";
        $db_password = "PASSWORD";
        $conn = new PDO('mysql:host=localhost;dbname=DATABASE', $db_username,
        $db_password);
        $conn -> exec("set names utf8");
        $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

        } catch(PDOException $e) {
        echo 'ERROR: ' . $e->getMessage();
        }
return $conn;
```

This method makes a **PDO** connection to the database and returns the connection to its caller. **PDO** is a new way of connecting to the database available since **PHP 5.1** and allows switching database a lot easier and painless since it works with several database drivers described below[18].

- CUBRID
- FreeTDS / Microsoft SQL Server / Sybase
- Firebird
- IBM DB2
- IBM Informix Dynamic Server
- MySQL 3.x/4.x/5.x
- Oracle Call Interface

- ODBC v3 (IBM DB2, unixODBC and win32 ODBC)
- PostgreSQL
- SQLite 3 and SQLite 2
- Microsoft SQL Server / SQL Azure
- 4D

4.2.5 Security

As said before, some redirections are pointed to multiple functions. In this case it was used for two ways of security verifications.

System Authorization allows the use of the *API* to only allowed devices, applications or system. This allows to control who has access to the *API* avoiding undesired traffic or misuse. This verification is called in every request to the *API* and it verifies a token or *IP* location to authenticate.

User Authorization allows access to certain information after verification of the user, in this case a student. This information is specific to that user like the curricula that a student is enrolled. When this type of requests are called the *API* will verify if a token has been provided. If it isn't it will deny the request. The application will then ask the user to authenticate using ISCTE-IUL credentials. After *LDAP* system confirms the authentication, the *API* will insert a token in the database and provide that same token to the user. In every request made by that user a token has to be provided and then compared to the database.

4.2.5.1 Token

To authorize the user and grant access to some requests a token needs to be provided to the *API*. This token works as a replacement for *username* and *password*. Below is an example of how the token is created and used.

FIGURE 4.1: Unauthorized request to the API

A user without a *token* tries to access information through a specific student request to the *API*. Since the token hasn't been provided, the *API* will reject the request and the application will ask the user to login using ISCTE-IUL credentials.

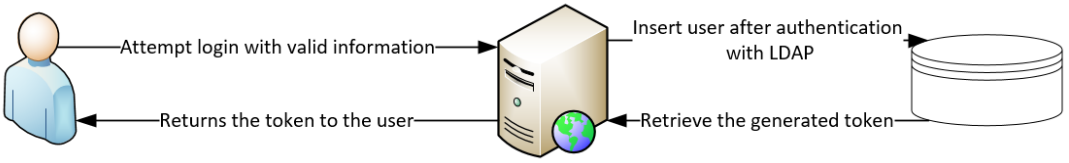

FIGURE 4.2: Login Attempt and token provided from API

The user then will try to login using his credentials. This information is received by the *API* which will verify against *LDAP*. If correctly it will insert the user in the database to generate a *token* and an expiration date using a *trigger* in the database. Then it will retrieve the *token* and return it to the user.

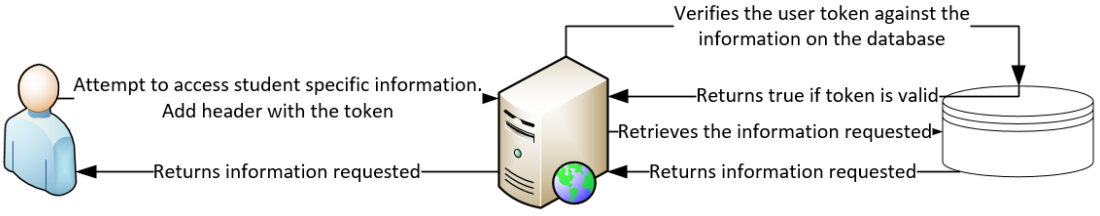

FIGURE 4.3: Unauthorized request to the API

In each request the application inserts the *token* in a header to authorize the request. The *API* will first verify the *token* and then make the request and return the information requested.

4.2.6 Code

After some experimentations with *Slim* framework it was possible to create the *Restful API* that is used by the application.

Below is an example of the request that returns all the teachers that lecture classes to a specific student. It is possible to verify all the code written that the framework will use from the received request to the response. Since it is a function about a specific user it will also be required a user authentication if a not is not provided. If it is not provided the user will have to authenticate like described in the section above.

```
$app->get('/teachersByStudent/:username', 'authenticate', 'authorizeToken',
 'getTeachersStudent');
```

This is the route that the request will use if called by **URL/teachersByStudent/:username** where **:username** is the student username. The route then tell us that it will call the **authenticate** function, **authorizeToken** function and **getTeachersStudents** function.

The **authenticate** function will verify if the system is allowed to use the application by verifying its token or IP location.

```
$headers = apache_request_headers();
$api_key = $headers['authorization'];
```

The **apache_request_headers** will obtain all the headers provided with the requests and it will be used to retrieve the token provided in the header **authorization**. After that it will call the **isValidApiKey** function.

```
function isValidApiKey($api_key) {
        $dbCon = getConnection();
        $stmt = $dbCon->prepare("SELECT id from `api_auth` WHERE api_key=:key");
        $stmt->bindParam("key", $api_key);
        $stmt->execute();
        $num_rows = $stmt->rowCount();
        $stmt->closeCursor();
        $dbCon = null;
        return $num_rows > 0;
}
```

This function will receive the *API* key provided and will be checked against the database information. If it is verified it will return **TRUE**, if not it will return **FALSE**. After this if it is **TRUE** it will continue to the next function, and if not it will send a response.

After the authenticate function, it will now call the **authorizeToken**.

```
$headers = apache_request_headers();
$token = $headers['bearer'];
$user = $route->getParam("username");
```

It works the same way as authenticate function, but now getting the **bearer** header. After retrieving the token it will get the **username** from the URL and then will call the **isValidToken** function and pass those two variables.

```
function isValidToken($user,$token) {
        $dbCon = getConnection();
        $stmt = $dbCon->prepare("SELECT * from `auth_token` WHERE token=:token AND
         username=:username AND date_expiration > now()");
        $stmt->bindParam("token", $token);
        $stmt->bindParam("username", $user);
        $stmt->execute();
        $num_rows = $stmt->rowCount();
        $stmt->closeCursor();
        $dbCon = null;
        return $num_rows > 0;
}
```

The function is very similar to the **isValidApiKey**, but will not check if both the **token** and **username** are valid and inside the expiration date. After all the verifications are checked it will call the **getTeachersStudents**.

```
function getTeachersStudent($studentUser) {

        $sql = "SELECT * FROM `curricular`,`enrollments` WHERE `id` =
         `enrollments`.curricular_id
        AND `enrollments`.students_username =:username";
        try {
                $dbCon = getConnection();
                $stmt = $dbCon->prepare($sql);
                $stmt->bindParam("username", $studentUser);
                $stmt->execute();
                $curriculars  = $stmt->fetchAll(PDO::FETCH_OBJ);
```

After the function header which declares that a **$studentUser** is used as a parameter a *SQL* query is written to a variable, which will be used after connecting to database to prepare a statement. Prepared Statements prevent *SQL* Injection to the database to ensure a more secure use of the data given by the *API* user. Then it will bind the parameter **$studentUser** to the **:username** in the *SQL* query, execute the query and then store all the objects inside the array to the **$curriculars** variable, which will have all the curricula for the specific student.

PHP does not require a variable declaration or type specification, so therefore the **$curriculars** variable was instantialized with the array.

```
if(!empty($curriculars)){
        $myteachers = array();
        $stringMyteachers = '';
        foreach($curriculars as $c){
                $sql_query = "SELECT * FROM `teachers` ,`professorship`
                WHERE
                `professorship`.teacher_username = `teachers`.username
                 AND `professorship`.curricular_id=:curricularId";
                $stmt = $dbCon->prepare($sql_query);
                $stmt->bindParam("curricularId", $c->id);
                $stmt->execute();
```

Then it will verify if any curricular was found, and if so it will iterate every curricular and make query to select all teachers that lecture that specific curricular, using prepared statement and binding **:curricularId** from **$sql_query** with the curricular id received in the first query. After that it will execute the statement.

```
                $teachers  = $stmt->fetchAll(PDO::FETCH_OBJ);
                $stringMyteachers=$stringMyteachers.'{"curricular":
                {"name":"'.$c->name.'","nameEn":"'.$c->nameEn.'",
                "teachers": ' . json_encode($teachers) . '}},';
        }

        $dbCon = null;
        $stringMyteachers = trim($stringMyteachers, ",");
        echo '['.$stringMyteachers.']';

}


} catch(PDOException $e) {
echo '{"error":{"text":'. $e->getMessage() .'}}';
```

```
        }
}
```

Finally it will fetch all teachers objects that lecture that curricular and concatenate it to the **$stringMyteachers** which will be return when all the **foreach** ends. If a **PDOException** is triggered, it will give a error message with the specific error.

4.2.7 Available Requests

To provide the information required to make the application work, several functions had to be created. Below is a list of all requests and responses.

/**login** Object with information about the logged in student and their specific token

/**students** Array with all students and information like email, name and username

/**students/:username** Object with information regarding a specific student

/**teachers** Array with all teachers and information like email, name and username

/**teachers/:username** Object with information regarding a specific teacher

/**teachersByCurricular/:curricularID** Array with all the teachers informations that lecture the specific curricular

/**teachersByStudent/:username** Array with all the teachers informations that lecture a curricular where a specific student is enrolled

/**curricularsByTeacher/:username** Array with all curriculars that a specific teacher lecture

/**curriculars/:username** Array with all curriculars that a specific student is enrolled

/curricular/:curricularID Object with information regarding a specific curricular

/events Array with all events

/events/:eventID Object with information regarding a specific event

/map/restauration Array with coordenates of the restauration places

/mytickets Returns the response provided by MyTickets API

/weather Returns the response provided by OpenWeatherData API

/strikes Array with all strikes

/strikes/:numberDays Array with the strikes for the following N days

/restaurant Array with all the restaurants

/meals/:restaurantID/:nr Array with the daily meals for a specific restaurant in the following N days

/map Array with all the Points of Interest (*POI*) in ISCTE-IUL with Latitude and Longitude

A more described table can be found in the appendix 7.

4.3 Mobile Application

After the development of the database and the *API* the only thing missing was the application itself. In this section, it will be possible to find all the related work for this application and all the knowledge needed to obtain over the first weeks with the framework and language.

4.3.1 Ionic

As defined in the concept phase, it was decided that *Ionic* framework would be the framework used to develop and build the application due to its current support and features it allows a powerful and fast hybrid application using only web technologies.

4.3.2 AngularJS

This project was the first time using *AngularJS*, although its complexity it has been proved to be an awesome framework.

AngularJS is a javascript framework created and maintained by *Google* and the community itself. It's an Open-Source with MIT license that helps the development of *Web Apps* also known as single-page applications. It provides some features like Data Binding and use of controllers[13].

Ionic uses *AngularJS* as its core programming language making essential to the development of the application[10].

4.3.3 Development

To start off developing on Ionic it is required its major dependency, which is *Cordova*. Both *Ionic* and *Cordova* can be installed using *node.js* package ecosystem known as *npm* which is the "the largest ecosystem of open source libraries in the world" as stated from node.js website[26].

After concluding the installation of *node.js, Cordova and Ionic* can easily be installed using one simple command from the terminal of the OS.

```
$ npm install -g cordova ionic
```

And then just create the ionic project using *Ionic* command-line interface (CLI).

```
$ ionic start myApp TYPEOFPROJECT
```

The type of project can be one of the three:

blank Clean project with no template.

tabs Project with a tab template on the bottom of the screen. Creates multiple templates and its needed modifications to the javascript files.

sidemenu Project with a side template which allows navigation through a menu on the left. Creates multiple templates and its needed modifications to the javascript files.

Now to start the app it only need to execute the following command:

```
$ ionic build
$ ionic run
```

The project created by *Ionic* contains several folders, although most of them are core parts of the *Ionic* framework.

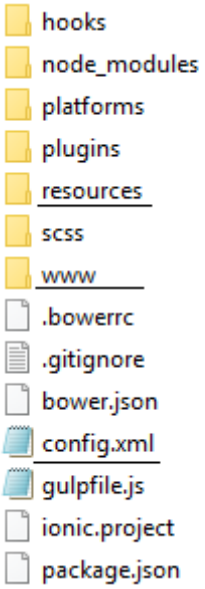

FIGURE 4.4: Struture of the project

In this project, the only main folders and file changed without the use of CLI were:

resources Splash image and Icon that the application will use.

www Code of the application, including Javascript and Cascading Style Sheets (CSS) used.

config.xml File used in the build of the app with the used resources, preferences and plugins used. Allows the generation of the manifest for the different OS.

Using the *CLI* it is possible to add platforms and plugins. Different platforms can be added to the same project, in this case *iOS* and *Android* has been added but *windows*, *blackberry* and several others can be added as well although *Ionic* is optimized for the first two. Plugins can be added through URL or manual installation. In this project all plugins have been added through URL using git repositories. Below are the *CLI* commands used.

```
\\ Adding platforms
ionic platform add android
ionic platform add ios

\\ Adding plugins
cordova plugin add URL
```

Each time splash and icon are changed, these changes need to be followed to the platforms and generated the OS specific images depending on their image guidelines. This is possible by doing:

```
// Both Splash and Icon
$ ionic resources

// Only Splash
$ ionic resources --splash

// Only Icon
$ ionic resources --icon
```

Since *Ionic* uses web technology to create a hybrid application, the **www** contains all the files to allow it to happen.

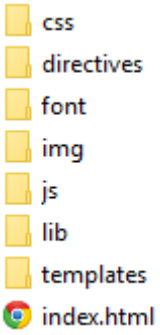

FIGURE 4.5: Struture of the www folder

This folder have a struture similiar to a website:

css Cascading Style Sheets files that define the style of web pages.

directives Angular directives that tells AngularJS HTML compiler which attachments or changes it has to do to a Document Object Model (DOM) element, like atributtes and names.

font File used in the build of the app with the used resources, preferences and plugins used. Allows the generation of the manifest for the different OS.

img Where all the images files that the application uses are stored.

js Javascript files including the core files app.js and controllers.js

lib Contains core files for AngularJS and Ionic

templates Template pages that will be compiled by AngularJS to provide the pages used in the application.

index.html Initial page that loads all Javascript files and Cascading Style Sheets for the application.

In the Javascript file named **app.js** every page also known as state is listed to be compiled using the information provided. Below is an example of declaring a state.

```
$stateProvider

.state('app.login', {
url: "/login",
views: {
'menuContent': {
                        templateUrl: "templates/login.html",
                        controller: 'LoginCtrl'
                                }
}

})
```

All states are added to the **$stateProvider** service. Each state contains the information of the URL, the controller that will manage it and the template that will be compiled.

When coding a app in *AngularJS* we need to take a look at their powerful features and how they affect the development. One of the key features of *AngularJS* is the use of data binding, allowing the application to synchronize the data between the model and view component.

An example used in the application that the user can see this feature is on the search of *POI*. When the user types to filter the results, it will automatically filter while the user type. This is possible due to the data binding.

The code bellow shows how this is possible.

```
<input type="text" placeholder="Pesquisa (Minimo: 2 caracteres)"
ng-model="searchAllRooms" value="">
```

Using the input tag known in the HTML language, we can define the name of the variable using ng-model attribute added in *AngularJS*. Each time a modification is made to this input, the variable **searchAllRoom** is updated.

```
<ion-item ng-if="searchAllRooms.length >= 2"  ng-repeat="mRoom in filtermarkers =
(markers | filter:{title:searchAllRooms,type_id:2})" href="#/app/map/{{mRoom}}">

\\Output variable
{{mRoom}}
```

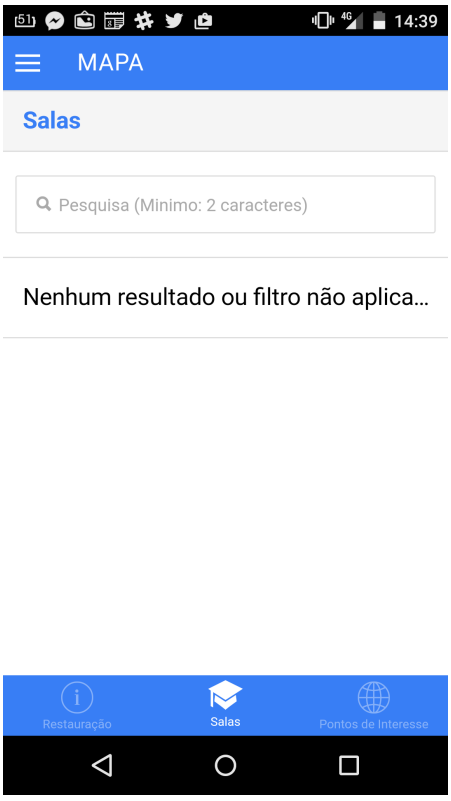

FIGURE 4.6: Map search with no filter applied

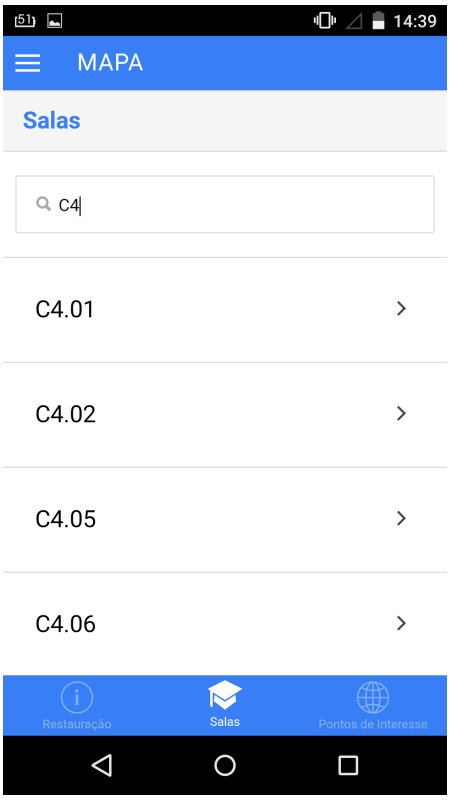

FIGURE 4.7: Map search with 'C4' filter applied

Ion-Item is a directive created by *Ionic*[11]. This directive is a child class of a list and will be used in a repeat with **ng-repeat** which is also added with *AngularJS*. This allows to list all the rooms within the filter applied.

The **ng-if** directive is similar to a if-else statement and in this case only allows to show rooms when a filter with 2 or more length is applied. This choice was to avoid a list with over 100 rooms to be listed which could cause a performance issue.

Finally, the **ng-repeat** creates a item per room, similar to a **for each** statement with a powerful potential due to its filter, allowing to filter by one or multiple variables.

To make this possible all the map markers have to be sent when the **Map** state/page is called. All markers are listed in a **JSON** response when the request **/map** is called to the *API*.

Although data bindings are very useful in *AngularJS* apps, in some cases a special attention is required to work.

A problem has been encountered in the early stage of the app when trying to pass the user information after login back to the page where the user was. For example, if the user tries to go see his curricula without being signed in, it will prompt the login page. After confirming the login it will go to the same page although the user variable **name**, **email** and **username** have not been updated. The reason for this situation is because the variables inside the login controller don't change the global variables. This has to be done in the main controller. To make this happen, an event trigger has to be created in the main controller.

```
$scope.$on('loggedIn', function(event, user){
        $timeout(function(){
                $scope.$apply(function(){
                        $scope.name = user.name;
                        $scope.email = user.email;
                        $scope.username = user.username;

                        localStorage.User_Name = user.name;
                        localStorage.User_Username = user.username;
                        localStorage.User_Email = user.email;
                        localStorage.token = user.token;

                        $ionicHistory.nextViewOptions({
                                disableBack: true
                        });
                        if($scope.lastState == null || $scope.lastState == undefined)
                        {
                                $location.path("/app/home");
                        }
                        else{
                                var state = $scope.lastState;
                                $scope.lastState = null;
                                $location.path(state.url);
                        }
                });
        });
});
```

On **loggedIn** event a function is called. Basically the function stores all the information passed by the *API* after authenticating the user into global variables

and local storage so the user doesn't get prompted to login every time he opens the application. After that it will redirect the page also known as state to the previous one before login. After confirming the login in the **login** controller a broadcast as to be made to trigger the **loggedIn** event.

```
$rootScope.$broadcast('loggedIn', data);
```

Most of the controllers make a request to the *API* to provide the information to the user. This information is called using **$http** service provided by *AngularJS* that facilitates the connection with remote servers, in this case the *API*.

```
var req = {
        method: 'GET',
        url: $rootScope.baseUrl+'curriculars/'+$scope.username,
        timeout : 5000
}
```

To be able to do this, a request array is created with the URL, method and timeout. The URL contains the request **curriculars**/ and passes the **$scope.username** variable which is the username of the logged in student. The request **method** indicates which action to be executed in the *API*. Finally the **timeout** variable defines how long will the application wait for a response.

```
$http(req).
success(function(data) {
        $scope.curriculars = data;
})
.error(function(data, status, headers, config) {
                window.plugins.toast.showLongBottom('Problemas na ligacao com o servidor');

});
```

To finalize, the code shows how to execute the request to the *API* by calling **$http** service and in case it is a success it stores the data to the **$scope.curriculars** and if it gives an error it will show a toast saying there is a problem with the connection to the server.

The only thing missing on the request is the system authorization and user authorization if required. These are passed in the HTTP headers.

FIGURE 4.8: Toast in case of an error

Authorization Token that authorize the device/system to use the *API*.

Bearer Token that authorize the user for certain *API* requests.

These tokens are passed using a interceptor available in **$http** service from *AngularJS*. The interceptor in each HTTP request pushes the tokens to the headers of the requests and forwards the request.

```
$httpProvider.interceptors.push(function($q, $location,$rootScope) {
    return {
            'request': function (config) {
                    $rootScope.$broadcast('loading:show');
                    if(config.url.startsWith($rootScope.baseUrl)){
                            config.headers.Authorization =
                            'SYSTEMAUTHORIZATIONTOKENHERE';
                            if (localStorage.token) {
                                    config.headers.Bearer = localStorage.token;

                            }
                    }
                    return config;
            },
            'responseError': function(response) {
                    $rootScope.$broadcast('loading:hide');
                    if(response.status === 401 || response.status === 403) {
                            $rootScope.$broadcast('tokenexpired');
                    }
                    return $q.reject(response);
            },
            'response': function(response) {
                    $rootScope.$broadcast('loading:hide');
                    return response;
            }
    };
});
```

When a request is executed using **$http** service the application enters in a loading stage, and a broadcast for **loading:show** is called to place a spinner indicating that the application is loading. After that, if the URL starts with the base URL of the *API* it will add the system authorization token in the **Authorization** header and if there is a user token it will be placed on the **Bearer** header.

If the response from the *API* is a error, the **responseError** will be called, hiding the loading spinner and in case it is an expired user token it will broadcast **tokenexpired**. This will make the main controller call the login state so the user can authenticate.

In case of success, it will hide the loading spinner by broadcasting **loading:hide** and return the response.

Similar to the filter used in the map markers, it is also possible to create custom filters[12]. In this app it was required to format the dates differently due to issues with *iOS* dates. In Android date formats like **'DD-MM-YYYY'** were accepted but in *iOS* an **Undefined Error** would occur. So the dates had to be filtered and changed to **'DD/MM/YYYY'**. It also allowed to show the day of the week using the same filters.

```
.filter("formatDate", function(){
        return function(input){
                input = input.replace(/-/g, '//');
                var date = new Date(input);
                var weekday = new Array(7);
                weekday[0]=  "Domingo";
                weekday[1] = "Segunda";
                weekday[2] = "Terca";
                weekday[3] = "Quarta";
                weekday[4] = "Quinta";
                weekday[5] = "Sexta";
                weekday[6] = "Sabado";

                var monthNames = ["Jan", "Fev", "Mar", "Abr", "Maio", "Jun",
                "Jul", "Ago", "Set", "Out", "Nove", "Dez"
                ];
                var result = weekday[date.getDay()] + ", "+ date.getDate()+" "+
                 monthNames[date.getMonth()];
                return result;
        }
```

```
})
```

The filter **formatDate** replaces the date inside input variable from '-' to '/' and creates a date variable as Date Javascript class to get the desired date format.

```
{{stk.start_date | formatDate}}
```

The variable stk.start_date will be changed using **formatDate**. Another date filter is used to convert *Unix* timestamp to the desired date format.

Along with the date filters some images filters were also created. When a user logs in, a username variable is stored to indicate which user has logged in. This variable is going to be used to get the image of the student from the *API*.

```
.filter("getImageUrl", function($rootScope){
        return function(input){
                var url = $rootScope.baseUrl+'images/'+input+".png";

                return url;
        }
})
```

When **getImageUrl** filter is called, it will return an URL with the base URL of the *API* and the rest of the path for the image.

4.3.4 Layouts

The design of the app is simple and with the help of *Ionic* CSS it provides, it looks and feels similar to a native app. Below it is possible to see the icon used in the application and the splash art when loading the application.

In the early stages the application looked pale, although over the time it began to be more pleasant using the institution colors. Below is a comparison side by side of the menu, canteen and curricular state.

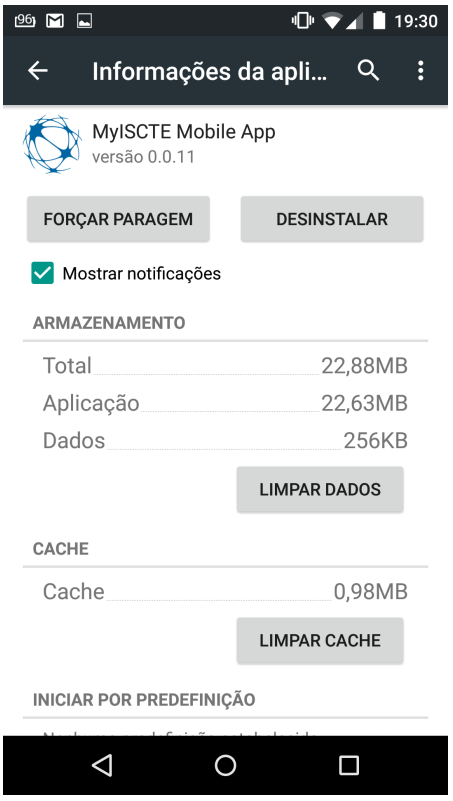

FIGURE 4.9: Icon used in the app

FIGURE 4.10: Splash screen used in the app

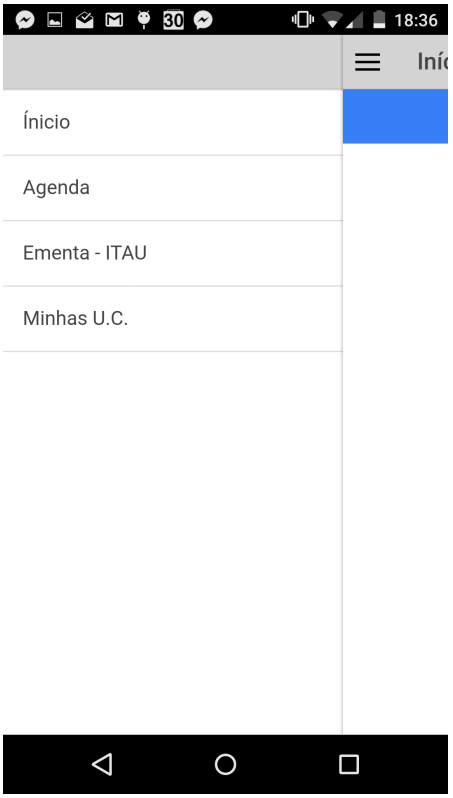

FIGURE 4.11: Old side menu

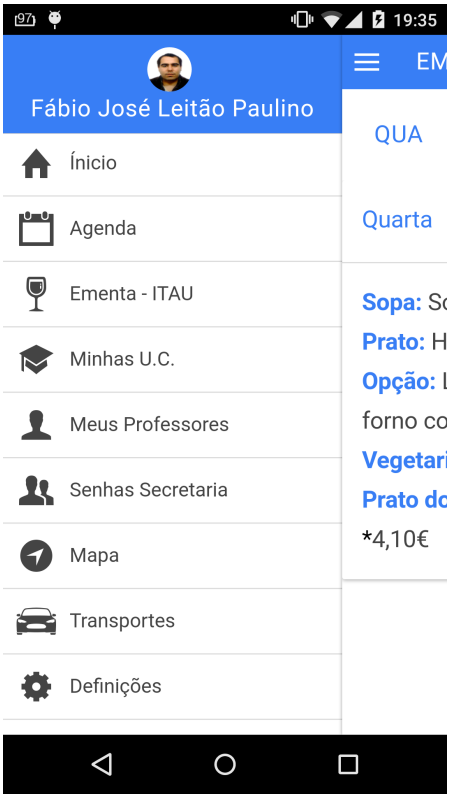

FIGURE 4.12: New side menu

The side menu can be opened with sliding from left to right, or simple pushing the three bar icon near the page title. The evolution from figure 4.8 and figure 4.9 is visible on the different information available as well as the header of the side menu. The new header allows to see if there is any student logged in and if it isn't, it will appear a button that will redirect to the login page.

A new set of colors, text font and icon were added to provide the user a more intuitive and simple to read menu.

FIGURE 4.13: Old canteen state

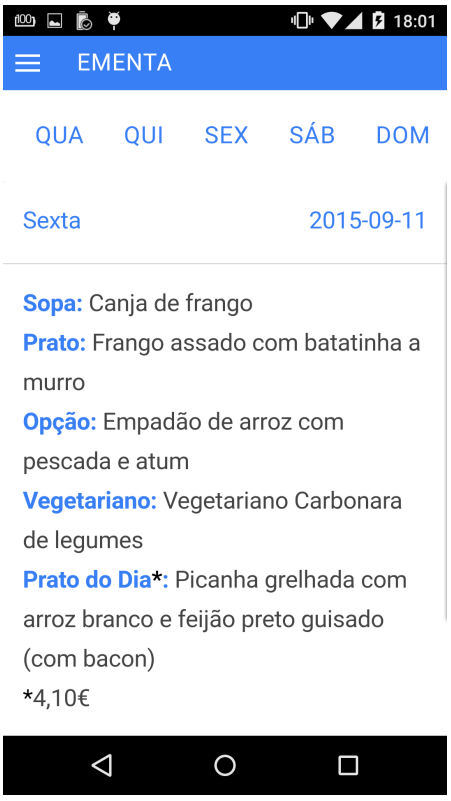

FIGURE 4.14: New canteen state

Initially, the canteen state presented the daily meals listed vertically and users had to scroll down until they found the desired day. In the figure 4.11 it is possible to see the visual overhaul that changed to a horizontal listing making users to scroll left or right to see their desired day of the week, which was also an improvement from only showing the day.

Finally the text font and size was improved for better readability and a faster interaction through the days was added at the top of the page.

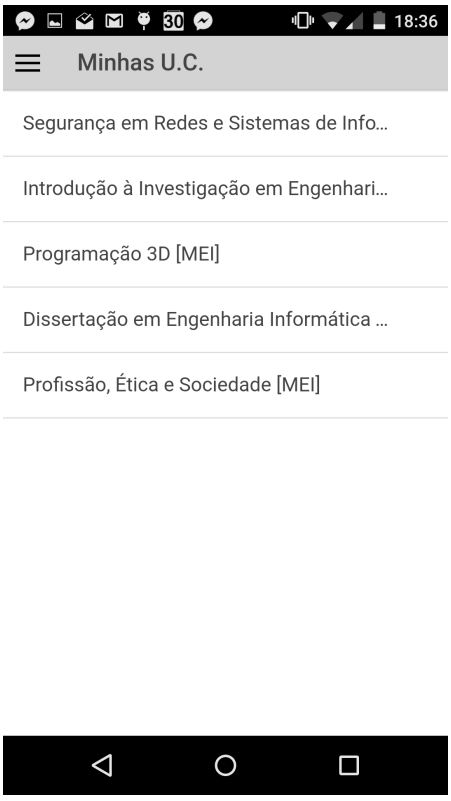

FIGURE 4.15: Old curriculars state

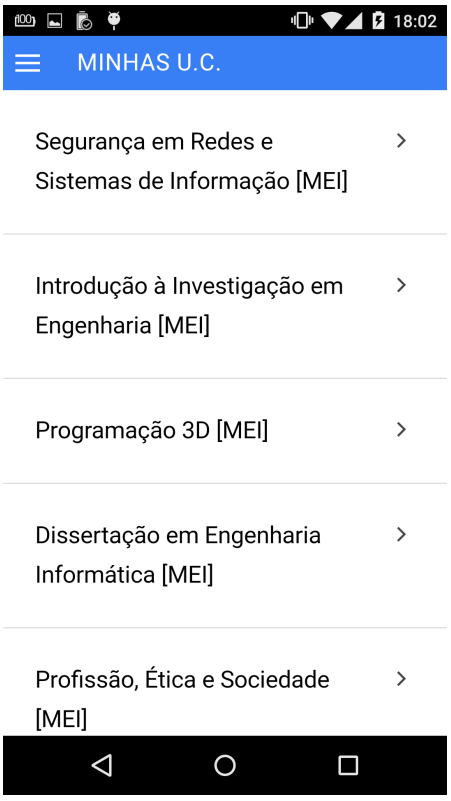

FIGURE 4.16: New curriculars state

The curricular state lists all the curricula that the authenticated user is enrolled. In the figure 4.12 it is possible to see that the state had a very simple layout with the curricular descriptions being cut due to the size of the text. Furthermore there was no reference to the user that by tapping in one of the curricular it would lead to another page. In figure 4.13 it is possible to see the changes made to the page that, similarly to the canteen page and menu, had the text font and size changed to increase readability and an arrow was added in the right side of each item (in this case curricular) to tell the user that something exists after tapping the curricular.

4.3.5 Limitations and Problems

As seen in the state of the art chapter hybrid applications don't support all *APIs* available in the mobile devices.

Android systems allow a creation of a service that runs in the background allowing the user to receive notifications regarding the app without having it opened. This feature could be used to notify the user about the tickets from *MyTicket API*. In *iOS* systems this feature would not be able to be implemented since the OS does not support applications or services to run on the background for a long period of time.

The most common alternative would be *Push* notifications. *Push* notifications are requests through an application server that will contact with the notification service which will forward the requested message to its target. The notification services are named **Apple Push Notification Service**[3] and **Google Cloud Messaging**[14]. *Push* notifications, also known as remote notifications, can be used using *Ionic* framework through a *Cordova* plugin.

One problem regarding the interface was the use of the **Materialize** *CSS*[1]. This *CSS* was created by a team of students from Carnegie Mellon University and although it is a powerful way to style websites to look similar to the *Material* design from *Google*[16] it was causing performance issues in some devices so it was removed in the later phase of the prototype.

Chapter 5

Results

To confirm the solution proposed in this project, several students were invited to test the prototype and check if their opinions meet the requirements. In this chapter it is possible to see the response from those students and in which real mobile devices the application was tested.

5.1 Students Tests

To increase the amount of testers, all the students tests were made in several computers from the institution using *Firefox* or *Chrome* browser. This decision was made due to the fact that the application runs in a *WebView* inside the device so the tests could also be made in the browser since there is no native mobile *API* that would impact the app.

A mobile simulation (see Image 5.1) was made using *iFrames* where the students selected the phone they wanted to test and size of the *iFrame* would be similar to the selected phone.

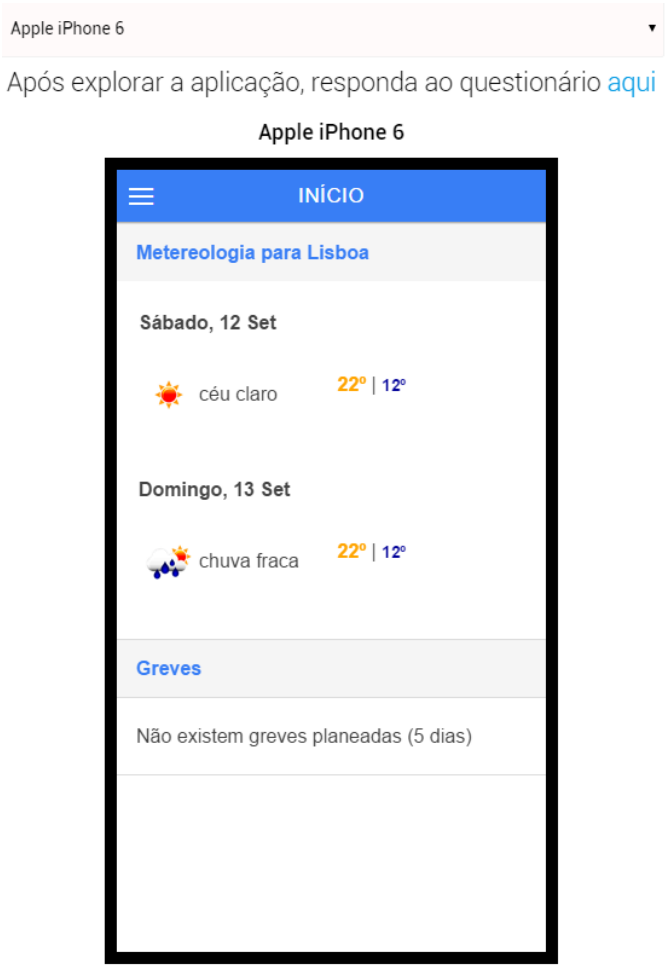

FIGURE 5.1: Page where students tested the app

In total, 66 students tested the application where the majority were male (83%) with ages between 21 and 25 years old. (see Image 5.2 and 5.3)

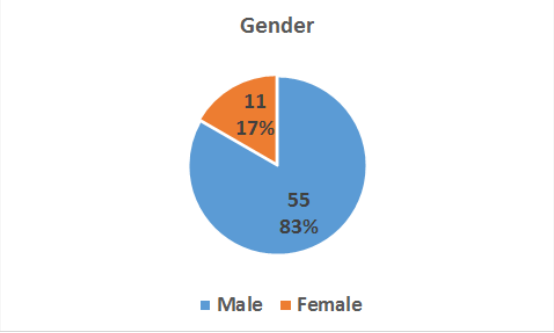

FIGURE 5.2: Chart with genre of the testers

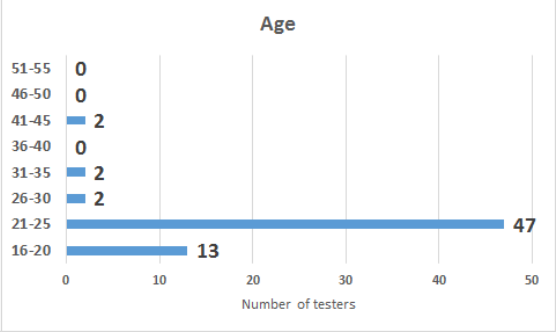

FIGURE 5.3: Chart with ages of the testers

Similar to the first questionnaire made in 2014, the majority of the testers are from engineering courses (see Image 5.4).

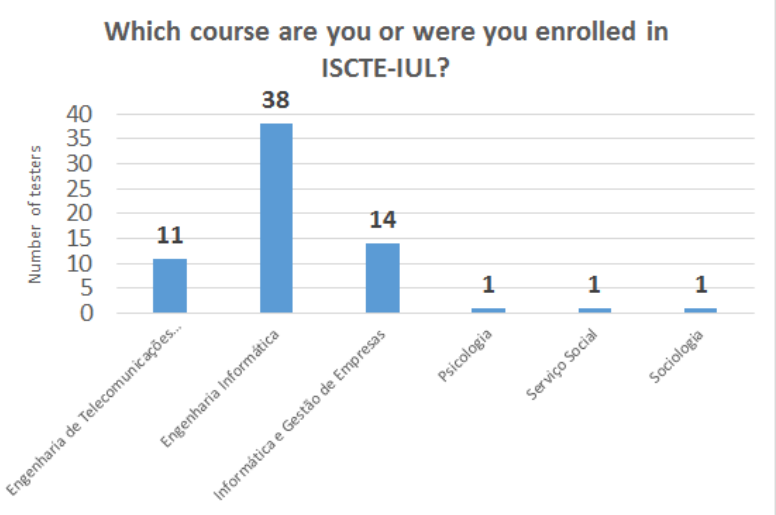

FIGURE 5.4: Chart with the testers enrolled courses

Only 1 *(1.5%)* of the 66 testers didn't own a smartphone (see Image 5.5). Regarding the often use of internet in their phones, 61 *(98.5%)* answered that they did use internet (see Image 5.6).

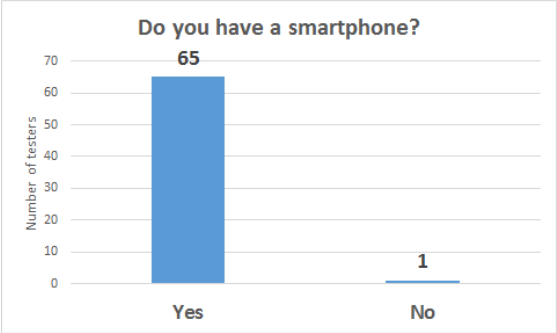

FIGURE 5.5: Chart with testers that have a smartphone

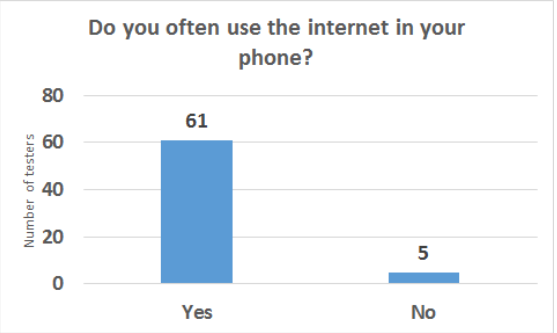

FIGURE 5.6: Chart with the internet usage of the testers

Most of the testers possess a *Android* smartphone *(72.7%)* followed by *iOS (18.2%)*, *Windows Phone (6.1%)* and *Blackberry (1.5%)* (see Image 5.7).

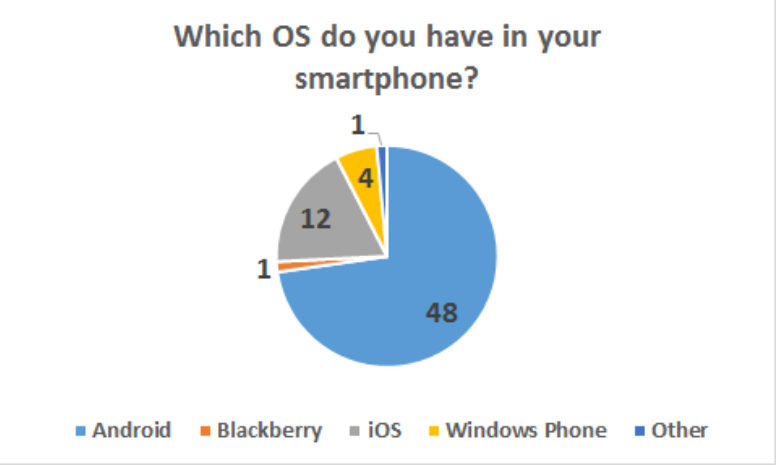

FIGURE 5.7: Chart with distribution on operative systems

Overall the testers rated the usability of the application at a average weight of 4,56 (see Image 5.8) and its utility at 4,76 (see Image 5.9) out of 5.

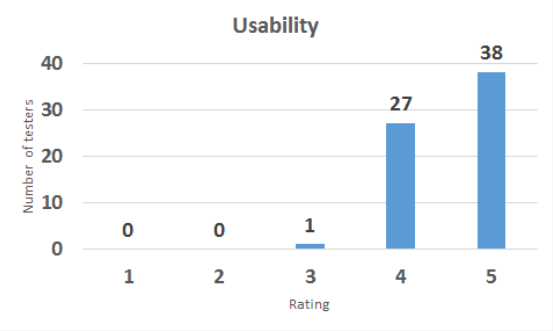

FIGURE 5.8: Testers opinion regarding usability of the app

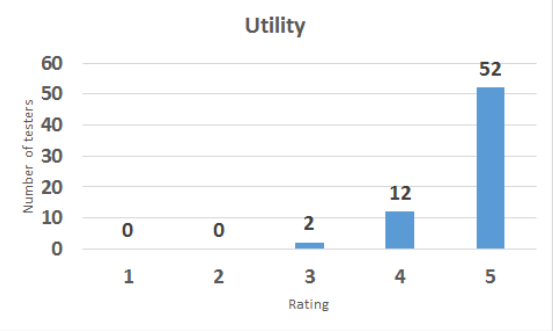

FIGURE 5.9: Testers opinion regarding utility of the app

In the general appreciation of the app the testers rated the application 4,52 out of 5 (see Image 5.10).

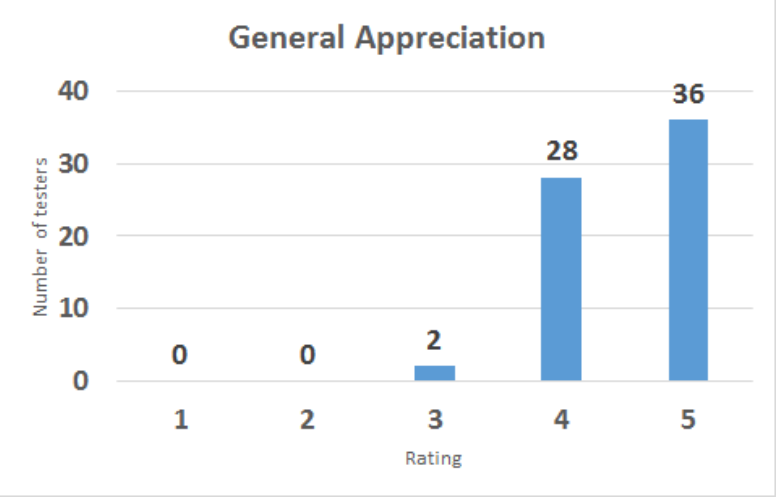

FIGURE 5.10: Testers opinion regarding the general appreciation of the app

Out of all the features, the testers considered that the worst feature with 3,88 out of 5 was the map, due to difficulty of seeing which place to go and the lack of *geo-location* of the user. The best one based on the testers evaluation was the ticket services with 4,74 out of 5 (see Image 5.11).

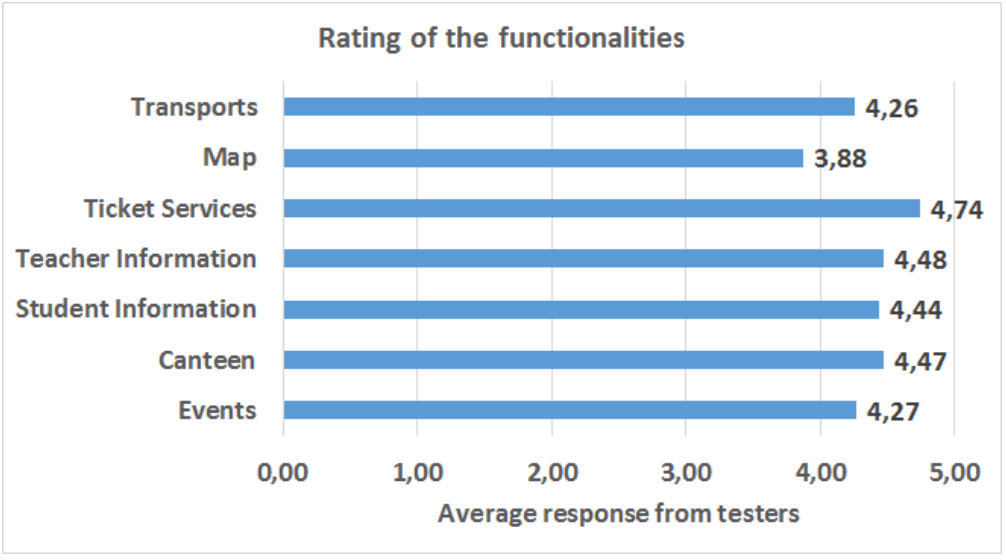

FIGURE 5.11: Testers evaluation regarding features of the app

All testers considered that the application was useful to their academic life in the institution (100%) (see Image 5.12) and that they would use the application (100%) (see Image 5.13).

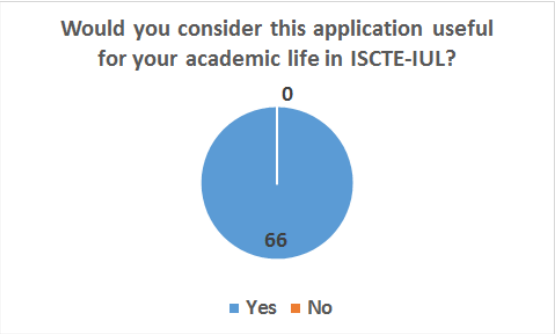

FIGURE 5.12: Testers opinion regarding usefulness of the app

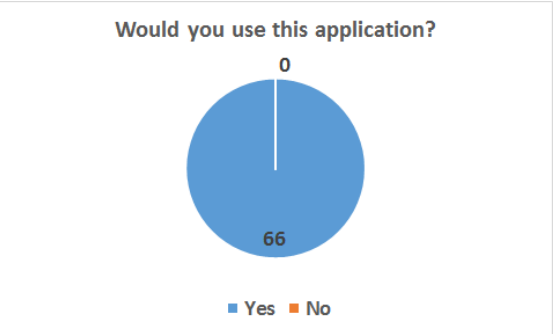

FIGURE 5.13: Testers opinion regarding if they would use the app

5.1.1 Suggested Functionalities

After the tests, the students were also queried regarding additional functionalities or improvements of the application. The majority of the testers wanted to see their private information regarding tuitions, grades, exams and schedule. This results were similar to the inquiry in October.

Although a map functionality is already implemented in the application, the students wanted an improvement to allow them to see the institution indoors. This feature could be implemented using Google Indoor Maps[15].

Other functionalities and improvements were suggested by the students and have been listed below.

- Integration with e-learning to allow students to see contents regarding their curricula
- More restaurants and meals information
- Contacts for the institution services
- Transports information regarding bus stops, trains and metro stations.
- Forum inside the application
- Advertising room rentals

- National News
- Library information

5.2 Real Devices

In addition to the student tests, some tests were performed using real devices. These tests helped checking for bugs with devices and *OS*. Two of those bugs were the date format problem between *Android* and *iOS* devices and the performance issue with the *Materialize CSS*. Both of these problems (and adopted solutions) were described in the previous chapter.

The *Android* devices were tested using the alpha testing provided by **Google Developer Console**[17]. Below are listed the devices used to test the application.

- Vodafone Smart 4 Turbo
- Sony Ericsson Xperia ray
- BQ Aquaris E4.5
- BQ Aquaris E4
- Google Nexus 4
- Google Nexus 5
- NVIDIA Shield Tablet
- LG G Pad 7.0"

Another operative system tested was the *iOS* using the *iPhone 6*.

Chapter 6

Conclusion

The results from the general opinion of the students towards the application met with the requirements obtained from the beginning of the project. Although some of the features that the students wanted couldn't be made due to the state of the technology available inside ISCTE-IUL the rest of the application had a great response.

The prototype made it clear that a mobile application is viable inside the institution and that the students are interested in it.

6.1 Future work

Unfortunately some features weren't possible to implement due to the information available or time until the prototype presentation. These are the features that were not possible to develop due to the lack of any information or support from ISCTE-IUL systems:

Schedule Allow students to view their schedule and classroom would help new and old students in the institution and would make it possible to point out where the classroom is.

Exams Students could see which, when and where would the exams be.

Tuitions An easier control for the students to know which tuitions they have overdue and which are still left to pay, with the necessary information to make the payment.

Map A better map with the interior of the institution would help newer students due to the complexity of the campus. Similar to other universities, a Google Indoor map would be a good addition to the application or institution.

At the end of the prototype presentation to the students, a new *API* was found called **One.Stop.Transport** that provides data regarding public transportation like schedules and stops[34]. This feature could be added to the application since most of the students use public transportation to get to and from the university.

And finally due to the change of the tickets system inside the institution a new adaptation is required to the new system.

Chapter 7

Appendix A

Request	Parameter	User Token	Response
/login	None	No	Object with information about the logged in student and their specific token
/students	None	No	Array with all students and information like email, name and username
/students/:username	Student Username	Yes	Object with information regarding a specific student
/teachers	None	No	Array with all teachers and information like email, name and username
/teachers/:username	Teacher Username	Yes	Object with information regarding a specific teacher
/teachersByCurricular/	Curricular ID	No	Array with all the teachers informations that lecture the specific curricular
/teachersByStudent/:us	Student Username	Yes	Array with all the teachers informations that lecture a curricular where a specific student is enlisted
/curricularsByTeacher/	Teacher Username	Yes	Array with all curriculars that a specific teacher lecture
/curriculars/:username	Student Username	No	Array with all curriculars that a specific student is enlisted
/curricular/:curricularI	Curricular ID	No	Object with information regarding a specific curricular
/events	None	No	Array with all events
/events/:eventID	Event ID	No	Object with information regarding a specific event
/map/restauration	None	No	Array with coordenates of the restauration places
/mytickets	None	No	Returns the response provided by MyTickets API
/weather	None	No	Returns the response provided by OpenWeatherData API

Request	Parameter	User Token	Response
/strikes	None	No	Array with all strikes
/strikes/:numberDays	Number days, e.g 3, for the next 3 days	No	Array with the strikes for the following N days
/restaurant	None	No	Array with all the restaurants
/meals/:restaurantID/::	Restaurant ID and number of days, e.g 3, for the next 3 days	No	Array with the daily meals for a specific restaurant in the following N days
/map	None	No	Array with all the Points of Interest (POI) in ISCTE-IUL with Latitude and Longitude

Bibliography

[1] Alex Mark Kevin Louie Alvin Wang, Alan Chang. Materialize - A modern responsive front-end framework based on Material Design. http://materializecss.com/. [Online; Seen on 12-September-2015].

[2] Esteban Angulo and Xavier Ferre. A case study on cross-platform development frameworks for mobile applications and ux. In *Proceedings of the XV International Conference on Human Computer Interaction*, page 27. ACM, 2014.

[3] Apple. Apple Push Notification Service. https://developer.apple.com/library/ios/documentation/NetworkingInternet/Conceptual/RemoteNotificationsPG/Chapters/ApplePushService.html. [Online; Seen on 11-September-2015].

[4] Jamie Appleseed. 8 Limitations When Designing For Mobile. http://baymard.com/blog/mobile-design-limitations, 2012. [Online; Seen on 06-January-2015].

[5] Andre Charland and Brian Leroux. Mobile application development: web vs. native. *Communications of the ACM*, 54(5):49–53, 2011.

[6] Jonathan Dann. Under the hood: Rebuilding Facebook for iOS. https://www.facebook.com/notes/facebook-engineering/under-the-hood-rebuilding-facebook-for-ios/10151036091753920, 2012. [Online; Seen on 16-January-2015].

[7] Universidade de Aveiro. UAMobile. http://www.ua.pt/stic/uamobile. [Online; Seen on 15-January-2015].

[8] Instituto Técnico de Lisboa. Técnico Lisboa Mobile. https://mobile.tecnico.ulisboa.pt/. [Online; Seen on 15-January-2015].

[9] Jolie Dell. Why LinkedIn dumped HTML5 & went native for its mobile apps. http://venturebeat.com/2013/04/17/linkedin-mobile-web-breakup/, 2013. [Online; Seen on 16-January-2015].

[10] Drifty. Ionic Framework. http://ionicframework.com/. [Online; Seen on 11-September-2015].

[11] Google. AngularJS - Directives. https://docs.angularjs.org/guide/directive. [Online; Seen on 11-September-2015].

[12] Google. AngularJS - Filters. https://docs.angularjs.org/api/ng/filter/filter. [Online; Seen on 11-September-2015].

[13] Google. AngularJS - HTML enhanced for web apps! https://angularjs.org/. [Online; Seen on 11-September-2015].

[14] Google. Google Cloud Messaging. https://developers.google.com/cloud-messaging/. [Online; Seen on 11-September-2015].

[15] Google. Indoor Maps. https://www.google.pt/intl/pt-BR/maps/about/partners/indoormaps/. [Online; Seen on 24-September-2015].

[16] Google. Material design. https://www.google.com/design/spec/material-design/introduction.html. [Online; Seen on 12-September-2015].

[17] Google. Use alpha/beta testing & staged rollouts. https://support.google.com/googleplay/android-developer/answer/3131213?hl=en. [Online; Seen on 12-September-2015].

[18] The PHP Group. PDO Drivers. http://php.net/manual/en/pdo.drivers.php. [Online; Seen on 12-September-2015].

[19] Clare Hopping. Hybrid Apps: The Mules of the Mobile Market. http://www.appmakr.com/blog/hybrid-apps-mobile-market/, 2014. [Online; Seen on 17-January-2015].

[20] IBM. Native, web or hybrid mobile-app development. ftp://public.dhe.ibm.com/software/pdf/mobile-enterprise/WSW14182USEN.pdf, 2012. [Online; Seen on 19-January-2015].

[21] ISCTE-IUL. Documentação da API Pública (Versão 1.3). https://ciencia.iscte-iul.pt/api/doc. [Online; Seen on 11-September-2015].

[22] ISCTE-IUL. ISCTE-IUL Tickets. http://myticket.iscte-iul.pt/. [Online; Seen on 11-September-2015].

[23] Filipe Cabecinhas João Neves, Carlos Fonseca. Hoje há greve? https://hagreve.com/. [Online; Seen on 11-September-2015].

[24] Rob Allen Slim Framework Team Josh Lockhart, Andrew Smith. Slim 3 Beta 1. http://www.slimframework.com/2015/07/03/slim3-beta1.html. [Online; Seen on 11-September-2015].

[25] Rob Allen Slim Framework Team Josh Lockhart, Andrew Smith. Slim a micro framework for PHP. http://www.slimframework.com/. [Online; Seen on 11-September-2015].

[26] Inc. Joyent. Node.js. https://nodejs.org/en/. [Online; Seen on 11-September-2015].

[27] Brian Klais. Research: How Many Apps Are in Each App Store. http://pureoxygenlabs.com/how-many-apps-in-each-app-store/, 2013. [Online; Update: 25-September-2014; Seen on 04-January-2015].

[28] Universidade Lusófona. Lusófona Mobile. http://www.ulusofona.pt/mobile.html. [Online; Seen on 15-January-2015].

[29] Salesforce marketing cloud. 2014 Mobile Behavior Report. http://www.exacttarget.com/sites/exacttarget/files/deliverables/

etmc-2014mobilebehaviorreport.pdf. [Online; Seen on 10-September-2015].

[30] UMass Boston News. UMass Boston Releases Mobile App for iPhones and Android Devices. http://www.umb.edu/news/detail/umass_boston_releases_mobile_app_for_iphones_and_android_devices, 2014. [Online; Seen on 15-January-2015].

[31] University of Illinois. UIS Mobile. http://www.uis.edu/apps/. [Online; Seen on 15-January-2015].

[32] University of Oxford. Oxford Mobile. http://m.ox.ac.uk/. [Online; Seen on 15-January-2015].

[33] Massachusetts Institute of Technology. MIT Mobile. http://m.mit.edu/about/. [Online; Seen on 15-January-2015].

[34] One.Stop.Transport. One.Stop.Transport - Construído para o futuro. Hoje. https://www.ost.pt/. [Online; Seen on 12-September-2015].

[35] OpenWeatherMap. Weather API. http://openweathermap.org/api. [Online; Seen on 11-September-2015].

[36] Optimus. Cross-Platform Framework Comparison: Xamarin vs Titanium vs PhoneGap. http://www.optimusinfo.com/blog/cross-platform-framework-comparison-xamarin-vs-titanium-vs-phonegap/. [Online; Seen on 10-September-2015].

[37] PhoneGap. http://phonegap.com/. [Online; Seen on 19-January-2015].

[38] PORDATA. Assinantes / equipamentos de utilizadores do serviço móvel em Portugal. http://www.pordata.pt/Portugal/Assinantes+++equipamentos+de+utilizadores+do+servico+movel-1180, 2014. [Online; Seen on 08-January-2015].

[39] Universidade Católica Portuguesa. Universidade Católica Portuguesa desenvolve APP MYCATÓLICA. http://www.ucp.pt/site/custom/template/

ucptplportalpag.asp?SSPAGEID=386&lang=1&artigoID=9601. [Online; Seen on 15-January-2015].

[40] Pedro Nogueira Ramos. Caso de estudo - questionários online. In *Desenhar Bases de Dados com UML*, pages 120–134. Edições Sílabo, 2006.

[41] Kelly Rice. Why Facebook Ditched its Hybrid App. http://www.kinvey.com/blog/3414/why-facebook-ditched-its-hybrid-app, 2014. [Online; Seen on 11-January-2015].

[42] Patrick Rudolph. Hybrid Mobile Apps: Providing A Native Experience With Web Technologies. http://www.smashingmagazine.com/2014/10/21/providing-a-native-experience-with-web-technologies/. [Online; Seen on 08-January-2015].

[43] stccorp. EasyWSDL2PHP. http://sourceforge.net/projects/easywsdl2php/. [Online; Seen on 11-September-2015].

[44] Harvard University. All Harvard Mobile. http://www.harvard.edu/all-harvard-mobile. [Online; Seen on 15-January-2015].

[45] Xamarin. http://xamarin.com/. [Online; Seen on 19-January-2015].

[46] Xamarin. Xamarin Costumers. http://xamarin.com/customers. [Online; Seen on 10-September-2015].