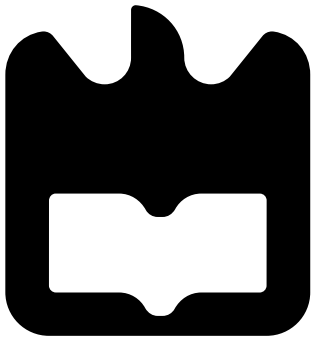
Universidade de Aveiro Departamento de Electrónica, Telecomunicações e Informática,
2016

Ricardo Filipe Ferreira Martins

Plataforma integrada para divulgação do estado da qualidade do ar

An integrated platform to access air quality levels



Ricardo Filipe Ferreira Martins

An integrated platform to access air quality levels

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requesitos necessários à obtenção do grau de Mestre em Engenharia de Computadores e Telemática, realizada sob a orientação científica do Professor Doutor Ilídio Castro Oliveira, Professor Auxiliar do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro e do Professor Doutor Carlos A. D. Soares Borrego, Professor Catedrático do Departamento de Ambiente e Ordenamento da Universidade de Aveiro.

o júri / the jury

presidente / president

Professor Doutor Joaquim Manuel Henriques de Sousa Pinto
Professor Auxiliar do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro

vogais / examiners committee

Professor Doutor Ilídio Fernando de Castro Oliveira
Professor Auxiliar do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro

Professor Doutor Ciro Alexandre Domingues Martins
Professor Adjunto da Escola Superior de Tecnologia e Gestão de Águeda

agradecimentos

Em primeiro lugar, gostaria de agradecer ao meu orientador, Professor Doutor Ilídio Fernando de Castro Oliveira pela sua disponibilidade, conhecimento e apoio prestados ao longo da realização desta Dissertação.
Gostaria também de manifestar o meu apreço ao Instituto do Ambiente e Desenvolvimento e a toda a sua equipa pela simpatia com que me receberam e por sempre demonstrarem a sua disponibilidade para o sucesso do projeto.
Agradeço por último a todas as pessoas que, no decorrer destes anos, me apoiaram e me permitiram crescer enquanto profissinal e ser humano, em especial aos meus amigos e à minha familia porque sem eles nada disto seria possível.

Resumo

Nesta dissertação é proposta uma solução que, partindo de dados recolhidos por equipamento instalado numa carrinha do IDAD, Instituto do Ambiente e Desenvolvimento, que é frequentemente deixada a realizar medições em locais de acesso inconveniente e que não fornece uma forma simples de acesso aos mesmos, permite disponibiliza-los para os membros do IDAD e para os clientes interessados.

A solução desenvolvida envolve todo o processo de fazer os dados chegarem ao utilizador, desde a sua recolha na carrinha, passando pelo seu envio para um servidor online através de um serviço de queueing de mensagens onde são alojados de forma a permitir o seu acesso em qualqer momento, tratamento dos dados obtidos e a criação de serviços REST para a sua disponibilizáção para serviços externos.

Além disso, foi criada uma interface web que permite aos utilizadores facilmente verificarem o estado da qualidade do ar em qualquer momento, controlar o acesso dos utilizadores e gerir a informação disponível para cada um. Por fim, foi também desenvolvida uma aplicação móvel Android que constitui o método mais simples e rápido de verificar o estado da qualidade do ar no momento atual e o qual abre a possibilidade de utilizar o sistema de notificações do Android para alertar o utilizador de situações de alarme e eventos específicos sem este ter que aceder manualmente a uma das plataformas para detetar essa situação.

Abstract

In this dissertation it is proposed a solution which, starting from data gathered by specialized equipment installed in a IDAD's vehicle, frequently left taking measurements in places of inconvenient access and which does not provide a simple way to access the data, allows the disclosure of them to the IDAD's team and all the interested clients.
The solution developed involves the whole process of taking the data to the user, since its collection, passing by its dispatch to an online server through a messaging service where they are stored for easy access at any time needed, data processing and the creation of RESTful services for their provision to external services.
Moreover it was developed a web interface that allows the user to easily check the state of the air quality at any moment, controlling user access and managing the data available for each one. Lastly, it was also developed an Android mobile application which represents the easiest and fastest way of checking the state of the air at the current time and also opens the possibility of using the notification system to alert the user of hazard situations and specific events without the need of manually accessing one of the platforms to detect that situation.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The constant growth of cities and industrialization leads to a prevalent problem in the environmental pollution, whether it is from factories' smoke releases, vehicles emissions or other sources. [1]

The impact of poor air quality is not only an health concern but also ecological and economical. Poor air quality can lead to the contamination of plants, animals, soil and water, which therefore leads to ecological and human issues, like the need for hospitalization or medical treatments, lost work days or reduction in the agriculture's productivity, for example, leading to a negative impact in the economy. Monitoring the air quality is, therefore, a fundamental practice to determine if the air quality in a region is within the expected values. [1]

Air quality monitoring aims at keeping track of a set of harmful gases in the atmosphere and determining if they are or not within a safe limit, according to the countries legislation. The goal is to give a perception of the state of air quality in a given place so that it is possible to take measures to enhance the air quality.

1.1 The Hosting Institute

Figure 1.1: The Institute's Logo

IDAD, Instituto do Ambiente e Desenvolvimento (http://www.ua.pt/idad/), is an association based in Aveiro, Portugal. It is a scientific and technical non-profit association, with public utility, funded in 1993, which works at the level of the environmental needs of Enterprises and Organizations.

The goal of the institute is to provide enterprises and public administration with the best and most innovative solutions aiming the environmental sustainability. Its relation to the University of Aveiro has allowed to achieve excellence results and recognition in the services provided.

The institute works in three main areas of intervention:

- Air Pollution
- Impact Assessment and Environmental Monitoring
- Sustainability

The work developed in this dissertation is inserted in two areas, Air Pollution and Environmental Monitoring. [2]

1.2 Motivation and Objectives

1.2.1 Motivation

A growing practice for monitoring and classifying air quality consists in continuously measuring the atmospheric concentration of a specific set of pollutants using sophisticated equipment, like specific gas analyzers. In Portugal, this is a common solution in fixed collecting stations spread across the country.

IDAD, however, offers a mobile, somewhat portable solution that allows for continuous real-time monitoring of the air quality at any location, rather than a fixed one. This service is used by some enterprises, like waste management companies or at places where the air quality state is of the public interest, like airports or highways. This alternative presented by the institute is based on three components: a vehicle, the LabQAr, reference equipment and a software, named Atmis.

The LabQAr (stands for *Laboratório móvel de monitorização da Qualidade do Ar*), is a mobile air quality monitoring laboratory. The Laboratory, which is built inside a van for mobility, is fully equipped with a set of analysers which continuously measure the atmospheric concentration of different parameters. The set of parameters measured and the methods used by the equipments are in accordance with the requirements imposed by law, making it a valid and trustable service for measuring air quality. It is also important to note that the equipment installed inside the van, which can be seen in Figure 1.2, are equal to the ones used in the majority of the Air Quality Measurement Networks in Portugal. [3]

All the sensors inside the LabQAr are connected to a desktop stored inside the van, through appropriate cables. This laptop has the Atmis software, which is responsible for periodically gathering the information from all the sensors, do some processing and store it in a local database. The software also provides an interface to access and analyse all the information about the measured air quality.

Figure 1.2: Environment sensors installed in the LabQAr

The problem with this solution is that the software is only accessible locally in the van and does not provide an out of the box solution to have access to it remotely. Currently, checking or retrieving the data stored in the local desktop requires to either have physical access to the vehicle, which is hardly the most practical solution, or to remotely connect to the vehicle's computer using a third-party peer-to-peer software, like TeamViewer, and manually access the information in the Atmis software. This means that neither the Institute or external companies have access to the information about the air quality state in real time. The Figure 1.3 shows an example of the LabQAr parked in an almost deserted area, next to an airport.

Figure 1.3: The LabQAr collecting at an airport

Source: https://www.ua.pt/ReadObject.aspx?obj=40391

1.2.2 Objectives

Motivated by the importance of the air quality monitoring and the need to address poor air quality condition as soon as possible, it is fundamental to provide the interested parties with a way of following the air quality as it evolves. The goal of this work is to conceive and implement a software platform, comprising a Website and a Mobile Application, to allow the Institute and its clients to access the data seamlessly, as it becomes available.

The solution should:

- Integrate user management to keep the data secure,
- Provide a web interface that allows the visualization of present and past air quality values, as measured in different field campaigns
- Provide a mobile application that allows real time access to air quality information anywhere
- Allow the configuration of alarms according to some events (data driven)
- Provide a user-friendly interface for an easy to use experience

1.3 Dissertation Structure

This dissertation is divided in seven chapters.

Chapter One presents the motivation and objectives that lead to the development of this work.

Chapter Two describes the State of Art of developing a full-stack solution, focusing on the different possibilities for web and mobile applications.

Chapter Three presents the main users of the system as well as the main functional requirements.

Chapter Four describes the system's architecture with emphasis on each of its components.

Chapter Five discusses the solution developed, explaining the most important details and how some of the features were implemented.

Chapter Six explains the tests conducted to validate the system.

Chapter Seven analyses the work completed as well as possible future work to improve and give continuity to the project.

Chapter 2

State of the Art

2.1 Air Quality Metrics

- **What is air quality**: Air quality is a term that is used to express a pollution level of the air. Air pollution has origin on many sources, both anthropogenic, which results from human activities, and natural causes.

  Air pollution is caused by a mixture of chemical substances in the air, that alters the natural constitution of the atmosphere. [1]

- **Reference Agencies**: The reference agency for air quality monitoring data in Portugal is the *Agência Portuguesa do Ambiente*,or APA, more specifically their QualAr (Air Quality) program. APA provides a website that is an online database about air quality. Their website aggregates data from different stations, each one managed by an Environmental Comission or Institute, like *Direção Regional do Ambiente e Ordenamento do Território dos Açores* or *Comissão de Coordenação e Desenvolvimento Regional do Norte*. [4]

  A typical way of representing the air quality metrics is by listing the concentration of each parameter at a certain place and time (as in the Figure 2.1) and the variation of a chosen parameter in the previous 24 hours (Figure 2.2)

Zona IQAr	Concelho	Estação	Tipo de Ambiente	Tipo de Influência	O_3	NO_2	CO	SO_2	PM_{10}	$PM_{2.5}$	C_6H_6
					máximo horário	máximo horário	máximo octo-horária	máximo horário	média diária	média diária	máximo horário
					µg/m³ às	µg/m³ às	µg/m³ às	µg/m³ às	µg/m³	µg/m³	µg/m³ às
Madeira/ Porto Santo	Santana	Santana	Rural	Fundo	72 24h	2 08h	-	-	11	3	-
Funchal	Funchal	São Gonçalo	Urbana	Fundo	65 14h	-	-	2 07h	8	1	-
	Funchal	São João	Urbana	Tráfego	-	63 09h	150 15h	-	10	8	-

'-' - Não Medido N.D. - Não Disponível

Figure 2.1: Latest values registered for each parameter at two different stations

Source: Values extracted from http://qualar.apambiente.pt/medicoes.grafico.php

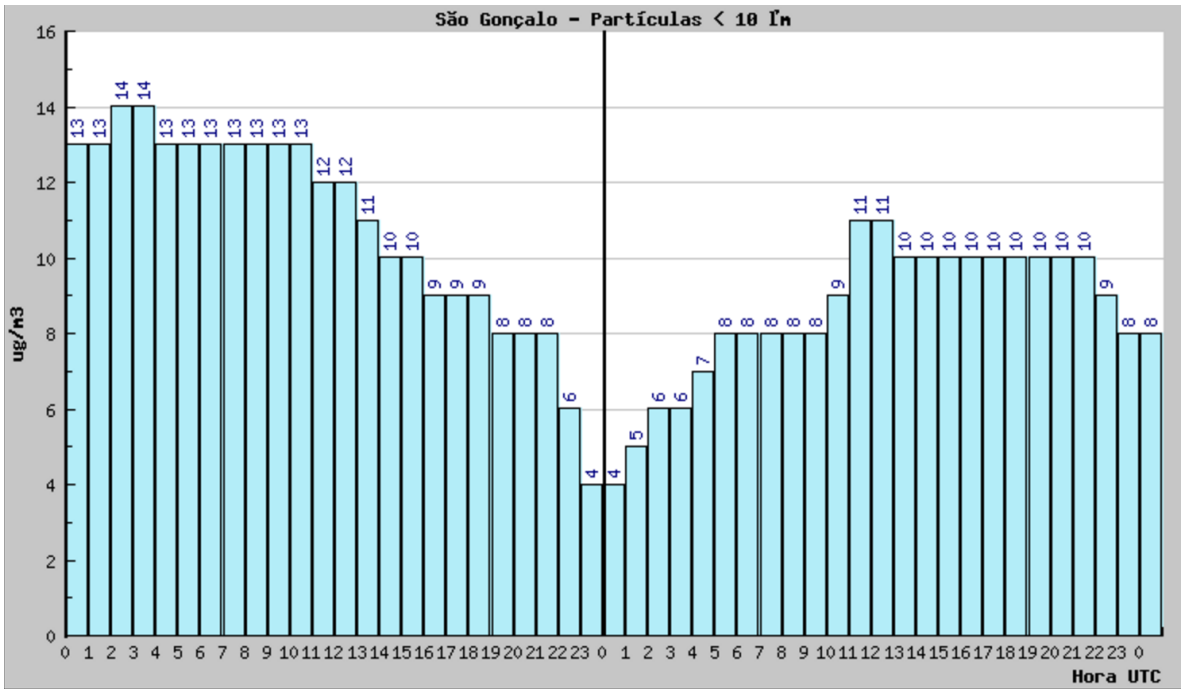


Figure 2.2: Variation of a parameter in the previous 24H

Source: Values extracted from http://qualar.apambiente.pt/index.php?

- **Air quality Index**: The air quality index is an easy way of translating the air quality that is easily understood both by people with a lot or none knowledge on the air quality topic.

  In Portugal this index is calculated by analysing five pollutants: Nitrogen Dioxide (NO2), Sulfur Dioxide (SO2), Carbon Monoxide (CO), Ozone (O3) and Particulate matter with 10 micrometers or less in diameter (PM10). The classification of each parameter is given by the chart in Figure 2.3.

Pollutant/ Classification	CO		NO2		O3		PM10		SO2	
	Min	Max	Min	Max	Min	Max	Min	Max	Min	Max
Bad	10000	-----	400	-----	240	-----	120	-----	500	-----
Weak	8500	9999	200	399	180	239	50	119	350	499
Moderate	7000	8499	140	199	120	179	35	49	210	349
Good	5000	6999	100	139	60	119	20	34	140	209
Very Good	0	4999	0	99	0	59	0	19	0	139

Figure 2.3: Classification chart for each pollutant (in g/m3)

The overall index classification is given by the pollutant(s) with the worst individual

classification.

2.2 Web Development

Full stack development means working both with back-end and front-end technologies, from setting up the database to the interface and everything in between. [5]

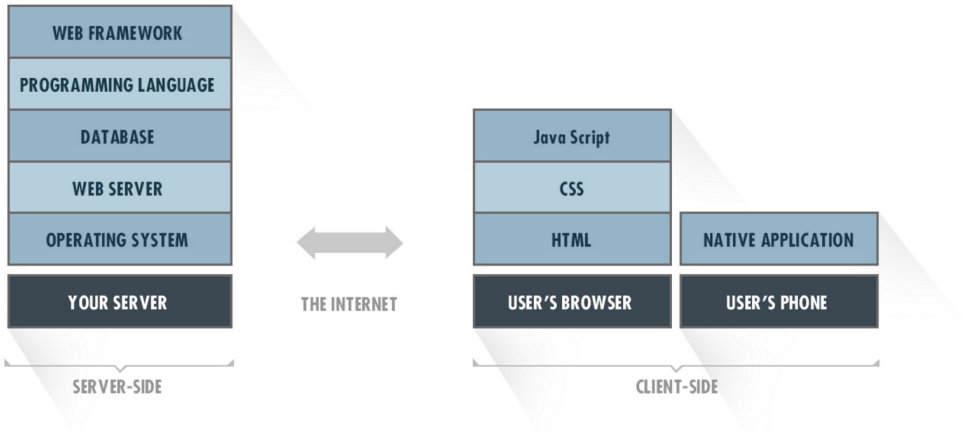


Figure 2.4: Layers of the full stack development

Source: http://svsg.co/how-to-choose-your-tech-stack/

Figure 2.4 shows the major blocks of a typical full stack system, in which each component builds on the features of the block below it. The server-side represents the back-end and the client-side represents the front-end.

Web development can be defined as a evolving set of languages that work together in order to receive, modify and deliver information using the Internet. Although this a concept simple to describe, the overwhelming variety of languages and platforms make it hard to implement, often requiring the developers demanding a broad and deep knowledge from the developer. [6]

When it comes to building a web application, the possibilities are nearly endless, both for back-end technologies and front-end technologies.

2.2.1 Back-end stack

The back-end containes all the logic that is never exposed directly to the user. All the information is passed and obtained to/from the user through the front-end stack.

The choice of a web framework to use also selects a programming language. Even though using a framework is not absolutely necessary, it allows to build an application that is structured and more maintainable, by resorting to a model that is tested and approved.

The choices available for the back-end stack are almost unlimited, but probably the most popular framework choices at the moment include:

- **Ruby on Rails:** Rails is a web framework that uses Ruby as its programming language, which is a general purpose programming language, best known for its use in web programming. [7] As the other frameworks, Rails defines a set of conventions that aid the maintenance of the applications and the collaboration between developers. The conventions are known as the Rails API which is documented and described in many books and articles. Rails combines the Ruby programming language with HTML, CSS, and JavaScript to develop the web application. [8]

  There are some advantages of using Ruby on Rails: it is an open-source framework and as such there are many third-party plugins available and well documented from the Rails community, freeing the developer from "reinventing the wheel"; it follows the MVC pattern [8], allowing for a good structure of the application; Ruby offers a clean syntax that is easy to read and write and has a big community with lots of guides and tutorials.

  Also, it is easy to develop small projects in Ruby on Rails very fast but the learning curve is steeper for bigger projects [9], which represents one of the framework's disadvantage. Another disadvantage of Rails is that it is considered to be one of the slowest frameworks, not being easily scaled.

- **Django:** Django is a web framework written in Python which focus on rapid development and clean, pragmatic design. Django combines the Python programming language with HTML, CSS, and JavaScript to develop the web application. [10]

  One of the advantages of using Django is that it offers a powerful database tool, the Django ORM, that handles the creation of the database, as well as insert, update, delete queries and other advanced querying.

  Also, similarly to Ruby, Python is a language easy to learn and to read and it is very easy to develop a simple application using the framework and follows the MVC pattern. Django has been around since 2006, and as such it has been highly improved over the years and also maintains by far the largest community out of all python frameworks who has built and maintained many powerful plugins.

- **Node.js:** Although Node.js [11] is not really seen as a web framework, it is highly used in the development of web applications. There are lots of web frameworks that are based in Node.js, like Express.js, Sails.js, Koa.js, Total.js, etc. [12] Node.js, in its core, is an open-source, cross-platform runtime environment for developing server-side Web applications. The majority of its modules are written in Javascript and the developers can use Javascript to develop new modules too.

  The paradigm behind Node.js is a bit different from Django and Ruby on Rails since it is focused in developing realtime web applications using push technology over websockets, while the others are more focused in the request-response paradigm. In Node.js the

applications have a two-way connection between the server and the client and either one can initiate the communication. In Django and Ruby on Rails the communication is always initiated by the client.

As well as the other frameworks, Node.js offers its own set of advantages and drawbacks. Like the two other frameworks, it is based on the open web stack, HTML, CSS, JS. Node is supposed to use non-blocking, event-driven I/O to stay light and efficient when faced with real-time applications. Node.js is capable of dealing with a lot of simultaneous connections with elevated throughput, making it useful for scalable network applications.

This exposes the frameworks' biggest weakness, the CPU intensive operations and other heavy computation. The other big issue with the framework is that it is not recommended to be used with relational databases, since the existing tools for relational databases for Node.js are still in an early stage. For this type of system, it is better use a solution like Django or Ruby on Rails. [13]

2.2.2 Front-end stack

When it comes to building the front-end, the stack of technologies is also subject of some branching although the choices here are not so important or restrictive. The basis of the stack is the HTML, CSS and Javascript. Additionally, it can be added some front-end Javascript frameworks, like AngularJS, BackboneJs or ReactJS but this is not a requirement. Also, it is possible to use presentation frameworks which provide a format for creating responsive web pages with clean aesthetics, like for example the Bootstrap or Google's Material Design Lite.

2.3 Mobile Computing

In today's world, mobile computing is an expression becoming increasingly familiar for both users and Developers. [14] As mobile devices keep evolving and becoming faster and cheaper and constant Internet connection becomes possible, mobile solutions have become almost an obligatory alternative for any company , either as a complete product or as a service. [15]

The improvements in the mobile devices performance, size and price, the almost constant access to Wireless or Mobile Networks make these devices a great alternative for quick access to the Internet and also as a way to support daily activities, and may even replace computers for some people. [16]

Starting around 2014, the number of mobile accesses to the Internet, using both smartphones, tablets, etc, as surpassed the number of desktop and notebook accesses, leaving open a huge market for mobile applications and solutions. [17]

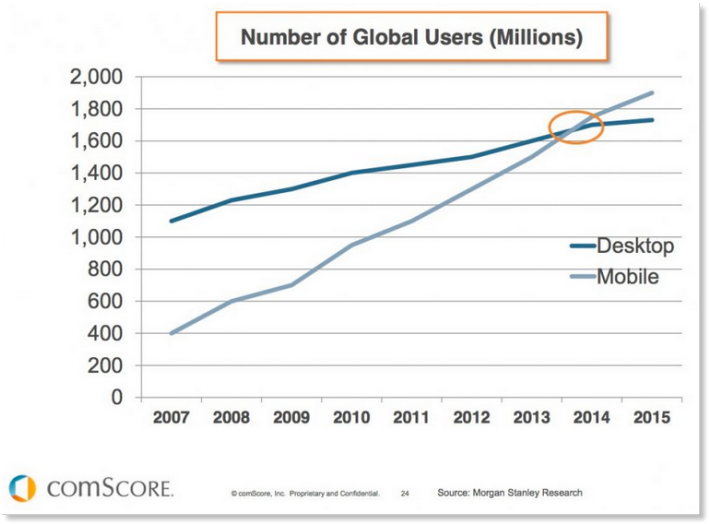


Figure 2.5: Number of users accessing the Internet through Mobile vs Desktop, in millions

Source: http://www.smartinsights.com/mobile-marketing/mobile-marketing-analytics /mobile-marketing-statistics/

2.3.1 OS Market Alternatives

The Mobile applications market is segmented into five main groups: Android, iOS, Windows Phone, Blackberry OS and all the others that do not represent a very significant part of the market [18]. While the last two segments have been losing ground to the others, and Windows Phone is still trying to gain some projection in the last years, it can safely be said that Android and iOS are the systems dominating the mobile market nowadays.

Given the fact that both of these architectures are dominating the smartphone market, adding to a share of 96.7% of the worldwide smartphones market (data from August of 2015), it is important to notice that Android wins the market by a clear advantage, owning almost 6 times more percentage of the market than iOS, with 82.8% versus 13.9% . [18]

Period	Android	iOS	Windows Phone	BlackBerry OS	Others
2015Q2	82.8%	13.9%	2.6%	0.3%	0.4%
2014Q2	84.8%	11.6%	2.5%	0.5%	0.7%
2013Q2	79.8%	12.9%	3.4%	2.8%	1.2%
2012Q2	69.3%	16.6%	3.1%	4.9%	6.1%

Source: IDC, Aug 2015

Figure 2.6: Worldwide segmentation on the smartphones market share, in percentage

Source: http://www.idc.com/prodserv/smartphone-os-market-share.jsp

Comparing the two main systems, Android's advantage over iOS comes primarily from the fact that Android is an open source mobile system while iOS is not, putting no artificial barriers to what the users can build and how they take advantage of the powerful mobile hardware. This opens the door for the production of Android devices from dozens of different companies, addressing hundreds of product lines varying from a large range of prices, features and quality adapting to each user's possibilities or desires, whereas iOS consist only in a specific set of devices falling into a specific price range, produced only by one company, Apple.

Also, Google Play offers an open marketplace for distributing the apps that can be used to easily distribute the application to all compatible devices. [14]

2.3.2 Native vs Web vs Hybrid

When it comes to building a mobile application, some decisions need to be made which can condition the features and the performance of the solution. Apart from the platform itself, the options for creating a mobile application fall into three different categories: Native Applications, HTML5 Applications or Hybrid Applications. [19]

- **Native Applications**: Native applications are built specifically for the platform where they will be used. These applications are installed on the device and make the best use out of its resources, having access to all its features, including camera, GPS, the notification system, accelerometer, multi touch features, faster graphics, etc. Unlike other solutions, native applications can be developed to work both online or offline, either fully or partially.

  However, this type of apps are also the most expensive to develop, requiring more effort and knowledge from the developers, specially if the application is needed across different platforms, which requires the developer to know all the platforms for which different versions of the application are maintained.

  Summing up, native applications offer better usability, better features and a better overall experience for the user, at the cost of most expensive and difficult development and maintenance.

  Native apps are made available to the users by placing them in the respective app stores of each platform, for example, the Google Play for Android or the App Store for iOS and can be easily installed from there, so they are very easy to find, install and use.

- **HTML5 Applications**: On the other hand, HTML5 Applications are, in fact, not true applications. They are simply a HTML5, Javascript and CSS3 web page, or a set of web pages, optimized to function in small displays and to mimic the feel and the flow of native applications. They follow a "write-once-run-everywhere" methodology making a single application available for all the platforms and devices, since they are open inside a web browser.

This type of applications lack the access to the native features of the device and always require Internet access in order to use it but, on the other side, updating the application is much easier because it only requires to update it once, instead of having to update it across every platform and making the update available on the respective store. One of the main problems of this approach is that devices have different screen sizes and resolutions, making the burden of testing the application across all the different screen sizes and resolutions fall on the shoulders of the developers.

HTML5 applications are fairly easier to develop, easier to provide support and can easily reach a wider range of devices making them a great solution for enterprises that need a simple product to be developed and made available to the largest number of users possible in a short time.

- **Hybrid Applications**: Hybrid Applications are a fusion between the HTML5 and the native approaches, being nothing more than a web application running inside a native wrapper. This wrapper, mostly, is simply a browser view that automatically connects to the web application URL. This simple change, gives access to the native features of the platform, opening doors to a whole new level of functionalities. Besides, hybrid applications are also obtained from the app stores and installed to the device making it easier for the user to get them.

  While the overall performance and user experience on this type of application is not the best, this is a good option when the application is needed on multiple platforms but it is still needed the access to some native feature like, for example, the notification system, or it is the company's interest to have the application available on the stores.

	Native	**HTML5**	**Hybrid**
App Features			
Graphics	Native APIs	HTML, Canvas, SVG	HTML, Canvas, SVG
Performance	Fast	Slow	Slow
Native look and feel	Native	Emulated	Emulated
Distribution	Appstore	Web	Appstore
Device Access			
Camera	Yes	No	Yes
Notifications	Yes	No	Yes
Contacts, calendar	Yes	No	Yes
Offline storage	Secure file storage	Shared SQL	Secure file system, shared SQL
Geolocation	Yes	Yes	Yes
Gestures			
Swipe	Yes	Yes	Yes
Pinch, spread	Yes	No	Yes
Connectivity	Online and offline	Mostly online	Online and offline
Development skills	ObjectiveC, Java	HTML5, CSS, Javascript	HTML5, CSS, Javascript

Figure 2.7: Comparison between the three mobile development approaches

Source: https://developer.salesforce.com/page/Native,_HTML5,_or_Hybrid:_Understanding_Your_Mobile_Application_Development_Options

Chapter 3

System Requirements

This section presents the main users of the system developed, as well as the main functional requirements indispensable to make it a useful service. It also gives a brief description of each requirement's function.

3.1 Actors

The system includes three different groups of actors. The first two groups represent the different types of users who have access to the same basic set of features, although one of them has access to some additional, more restrict features. These groups are the staff members and the partners. The last group does not represent a set of users itself but an already existing process.

- **Partners:** Partners represent all the users who will have access to the system's data and its main features, not belong to IDAD. These users are mainly employees from the companies who will use the system as a service and, as such, do not have access to private information and administration features.

  The features and type of data available to these users are the same, although each one will only have access to a specific set of data that is related to company he is associated to.

- **Staff Members:** The staff members are a vital piece to the system operation. They have access to all the features and data, including administrator's features, being the ones responsible for mapping each other user to his correct set of data and of maintaining some important data without which the system is not useful.

- **Environment Sensing System:** The last actor represents the part of the system responsible of collecting the main data required, obtained by the environmental sensors.

3.2 Usage scenarios

The requirements of the system can be divided into three segments, Website only requirements, mobile only requirements and website and mobile wide requirements. The first

segment encloses some features that are available only to the web application and the second encloses the ones that are exclusive to the mobile application. The last group encloses the set of features that are common for both platforms.

3.2.1 Website Use Cases

The Figure 3.1 illustrates the use cases diagram for the use cases that are specific for the Website, using the UML notation. [20] An additional analyses of the main use cases can be found in the Appendix section.

Figure 3.1: Website only use cases diagram

- **Check Air Quality Classification**: The user is able of checking the classification of the air quality in a chosen moment, as long as he has at least one campaign associated to him or he is a staff member.

- **Analyse Air Quality Variation In The Last 24h**: The user is able of examining the classification of the air quality during the previous 24 hours, as long as he has at least one campaign associated to him or he is a staff member.

- **Check Latest Data**: The user is able to check his most recent data. The result should be different according to the user. If the user is a staff member the latest data corresponds to the last data received from the car; if he is a partner the latest data corresponds to the last data received during his most recent campaign.

- **Analyse 24H Variation Of A Parameter**: The user is able of examining the variation of a parameter during the previous 24 hours, as long as he has at least one campaign associated to him or he is a staff member.

- **Analyse Variation Of a Parameter In A Campaign**: The user is able to check the daily variation of a parameter in the duration of a campaign, as long as he has at least one campaign associated to him or he is a staff member.

- **Select Campaign**: The user is able to select a campaign, as long as he has at least one campaign associated to him or he is a staff member, and access air quality metrics collected during that specific campaign. A campaign is a period of time during which the data is collected for a specific company.

- **Browse A Specific Date**: The user is able to select any date, as long as he has at least one campaign associated to him and selects a day during the campaign or he is a staff member, and access air quality metrics collected at that specific date.

- **Check detected errors and alarms**: The user is able of checking the past errors and alarms. The type of errors and alarms available varies according to the type of user.

- **Authenticate**: The user is able to login and logout of the system, as long as he has a valid account.

- **Manage Account Data**: The user is able to edit his account name, user name or password.

- **Manage Data Collection Campaigns**: The user is able to add, edit, or remove a period of time for which the car is collecting data in a location.

- **Manage Data Collection Points**: The user is able to add, edit, or remove a location for data collection to/from.

- **Manage Alarms Limits**: The user is able to add, edit, or remove a threshold value for each parameter, above which it will be generated a push notification.

- **Manage Users**: The user is able to add, edit or remove an user to/from the system.

3.2.2 Mobile App Use Cases

The Figure 3.2 illustrates the use cases for the Mobile application.

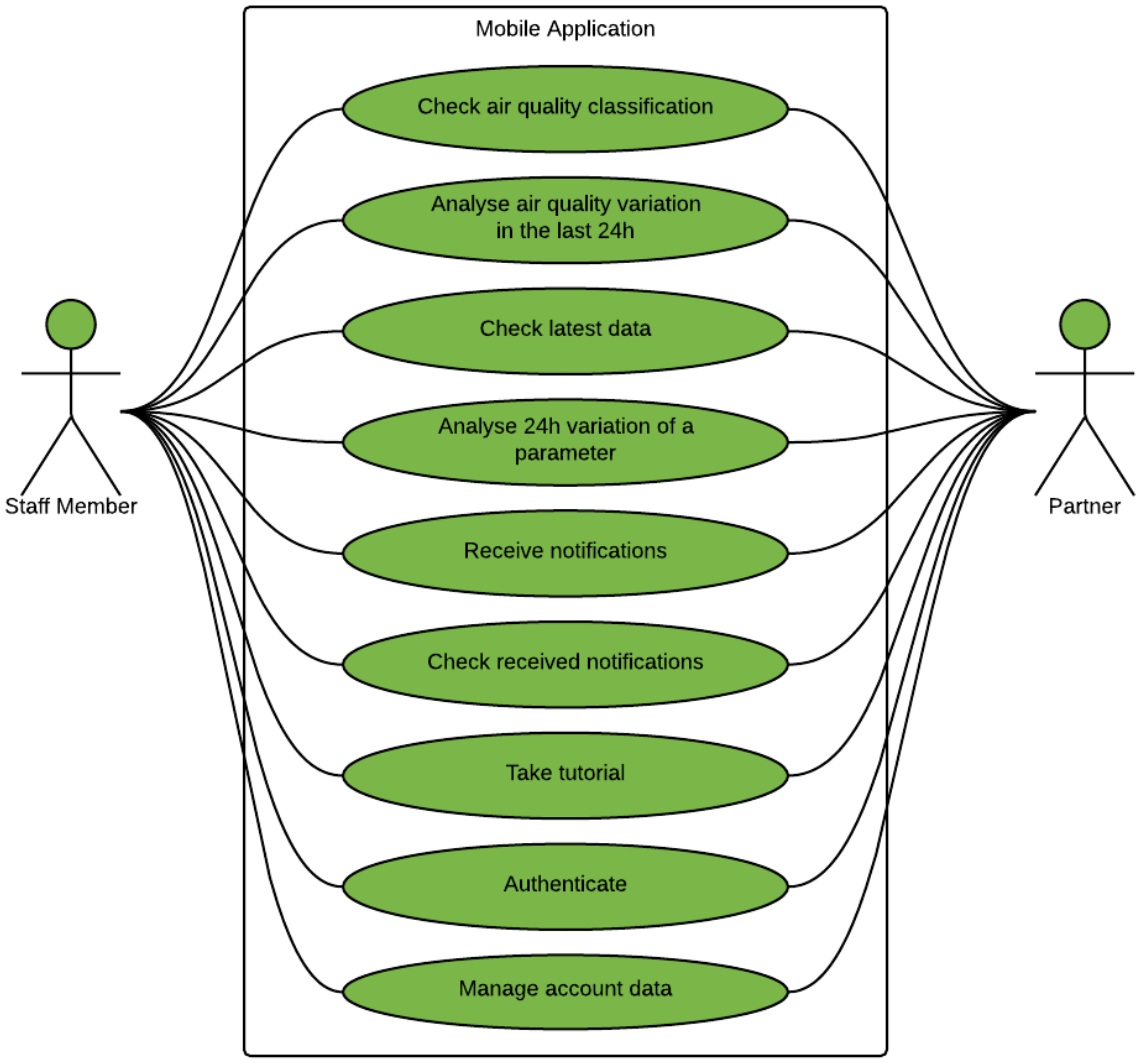


Figure 3.2: Mobile only use cases diagram

- **Check Air Quality Classification**: This use case is equivalent to the one specified for the website.
- **Analyse Air Quality Variation In The Last 24h**: This use case is equivalent to the one specified for the website.
- **Check Latest Data**: This use case is equivalent to the one specified for the website.
- **Analyse 24H Variation Of A Parameter**: This use case is equivalent to the one specified for the website.
- **Receive push notifications**: The user is able to receive push notifications on his

mobile device, as long as he has at least one campaign associated to him or he is a staff member.

- **Check received notifications**: The user is able of checking his past received notifications. The type of errors and alarms available varies according to the type of user.
- **Take tutorial**: The user is able of taking a tutorial the first time he uses the mobile application and, after the first utilization, anytime he desires.
- **Authenticate**: This use case is equivalent to the one specified for the website.
- **Manage Account Data**: This use case is equivalent to the one specified for the website.

Chapter 4

Architecture

In this chapter the developed architecture is presented. The complete flow of the system is explained, from getting the data from inside the vehicle until it is presented to the final user and for each step of the flow it is made a small description of the process involved. The solution develop is based on the system already existing in the LabQAr which means it is restrict to the problem presented before but it can be adapted/extended in the future.

4.1 LabQAr's Legacy System

The process starts inside the LabQAr, responsible to collect data from the sensors installed inside the van.

The data is collected by each sensor individually and it is aggregated by the Atmis software, installed in the computer inside the vehicle. The Atmis is a system for collecting and processing meteorological data and air data, based in standard protocols and interfaces developed specifically for air quality and meteorological data management. [21]

The software uses a local database, built using Windows SQL Server, to store the data from the sensors. This database is the starting point for the developed system's data flux.

This part of the architecture is very restrict and specific and it can not be changed.

4.2 System Architecture

The system architecture can be divided into five main components distributed along multiple platforms and devices (Figure 4.1). These components are the LabQAr, a RabbitMQ Server, a Main Server, a Web Application and a Mobile Application. In the figure, the components shaded with a blue color are the part of the solution that already exists, inside the van. The only modification there is the inclusion of a module that uploads the data from the sensors to the main server. As it can be seen, the uploaded data is sent through a TCP connection to a RabbitMQ Server before reaching the main server. There, the downloading module is installed, along with the global database and all the logic that supports both the Web and Mobile Applications. The Web Application interacts with the main server through HTTP requests whereas the Mobile Application uses a REST API.

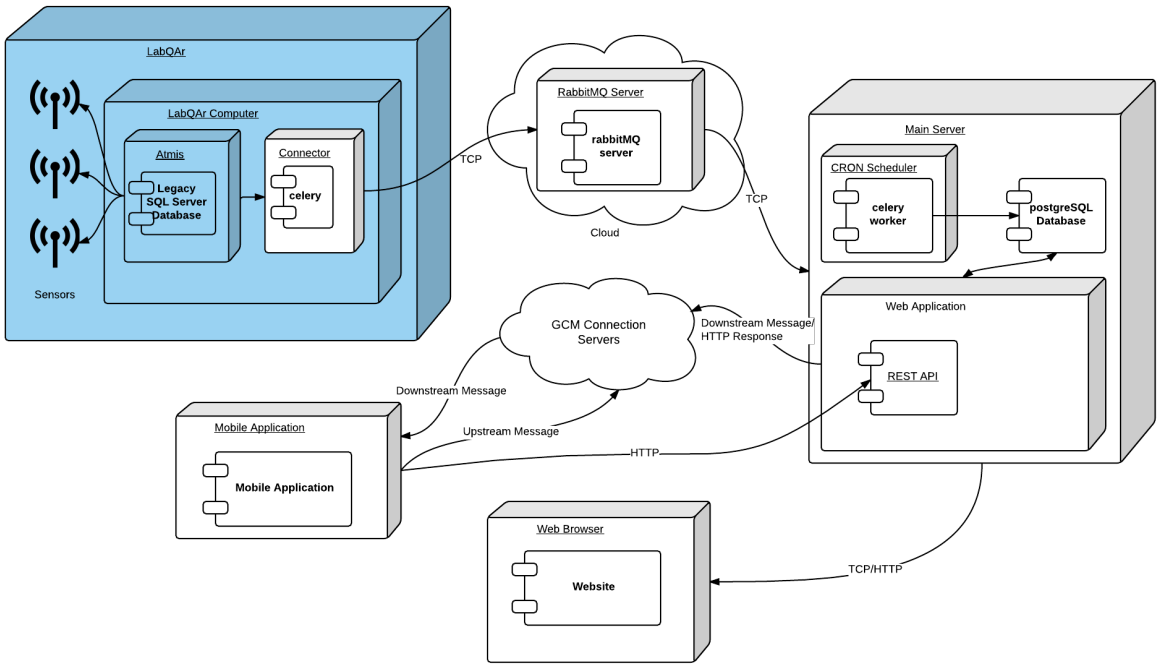


Figure 4.1: Proposed architecture

4.2.1 Watchdog in the LabQAr

In the LabQAr's computer we deployed the Connector module, which checks the local database for new entries and schedules them to be sent to the online database in the main server, in the form of Celery tasks. Celery is an asynchronous task/job queue based on message distribution which uses a message queue server for transporting the tasks to the main server. [22]

This Connector is executed by a Windows scheduled task that is set to run every hour. The reason behind the choice of making the task hourly comes from the fact that the Atmis software only saves the data from the sensors to the local database on an hourly basis.

4.2.2 RabbitMQ Server

The proposed architecture requires a messaging queue, working together with Celery to exchange the task messages between the LabQAr's computer and the Main server. The messaging queue can be implemented in a RabbitMQ Server inside the main server but since the main server is inside the Openshift Cloud, the queue is also inside a cloud service.

For this system, it is used RabbitMQ as the message broker, using the CloudAMQP service, which provides a highly available message queue on the cloud, making the bridge between the celery tasks created in the LabQAr and the main server, where the main module of the architecture exists. The tasks sent from the Connector in the van are stored temporarily in the cloud server, until they are fetched by the main server.

4.2.3 Main Server

The main server is built under the Red Hat's Openshift Cloud service which provides a hosting platform for the core of the architecture. Here lives the online database, a Celery Worker, the website and its web services.

The database used in the main server is a PostgreSQL database. PostgreSQL is a relational database, like the SQL Server used locally in the LabQAr, although it is open-source and can be used in all major operating systems while SQL Server only works on a Windows platform. The use of PostgreSQL allows, therefore, to keep the main server logic independent from the platform where it is installed. This database contains a replica of the all the tables and data existing in the offline database used by the Atmis software in the car, plus some additional tables required to store data relevant for the Web and Android Applications.

Another crucial piece of the architecture is the Celery Worker. To complement the Windows shared task scheduler used in the LabQAr, on the main server it is used the CRON jobs scheduler to periodically run a Celery Worker, which will check the CloudAMQP server for new tasks created by the Connector in the LabQAr and execute them locally. Each task executed by the Celery Worker represents an update to the main server's database.

This job could also be executed once every hour but, since the Openshift platform only support CRON jobs to be run every minute, hour, week or month, it could happen that the tasks were scheduled in the car at a given time and the CRON scheduler only start, in the worst case, one hour later. Therefore, using minutely CRON jobs is the best way to guarantee that the data on the server's database is updated as soon as its sent from the car.

4.2.4 Web Application

The web application is the last component inside the main server. The application is built using the Django Web Framework, which is a high-level Python Web framework that allows for rapid development and clean design.

This is probably the most important component of the overall architecture since it is responsible for managing user accounts, process the raw data so it can be shown to the user both on the web application and the mobile application, provide an administrator interface to manage the additional data needed across both platforms and also provide a REST interface to use the system from the mobile application. The web application is also responsible for checking the data for alarm situations and specified events and send them to the interested users in the form of notifications.

The Django application makes use of native Django libraries for managing user accounts and log in/log out features. For the REST API, it uses the Django REST framework, which is a powerful and flexible toolkit for building Web APIs. Lastly, the application uses the Django Push Notifications library, which connects to a Google Cloud Messaging (GCM) server to allow sending push notifications to the user's devices.

4.2.5 Mobile Application

Since Android is the most used mobile Operation System, as it was shown in Chapter Two, and since non-android users can still use the system through the Website, this is the system chosen for the mobile application.

The mobile application interacts with the rest of the architecture through the RESTful web services provided by the Django web application.

This application, similarly to the web application, is connected to a GCM server, so it can be listening for new push notifications from the server.

Chapter 5

System Implementation

The goal of the system is to provide a way of accessing the data from the LabQAr's equipment without needing to remotely or physically accessing the vehicle to do so, like the solution previously in use. The system does not intend, at any point, to fully replace the current solution, since the Atmis software offers some features that can not be covered in this dissertation, either for limited time or lack of specialized knowledge in the Air Quality field of study.

This chapter explains the implementation process of each component of the system, explaining in more detail some the key parts of the implementation.

5.1 Backend

The system is built around the data collected at the vehicle. The car doesn't have a stable network connection, which means it was not an option to provide an interface for accessing the data on demand, since the connection service level would vary considerably. This means the best option was to copy the data from the LabQAr to a more reliable location. The car uses a SQL Server database, with more than 30 tables, that was fully replicated to a cloud-based PostgreSQL database. Since the web application was built with the Django Framework and the application is necessarily connected to the database, it is possible to built the new database with the help of Django models.

In Django, a model represents the source of information about the data, since it contains the essential fields and behaviors of the data being stored. Usually, a Django model is mapped to a table in the database. Each model is defined by a Python class that extends **django.db.models.Model** where each attribute of the model represents a table field. By doing this, Django creates an automatically-generated database-access API that can be used for making queries inside the web application.

The following excerpt of code shows the declaration of a model for the table "Local" (Figure 5.1), which has six fields: an id, which is the primary automatically incremented by the model; a user, which is a foreign key that maps to an entry of another Django Model called "User" and some other simple fields.

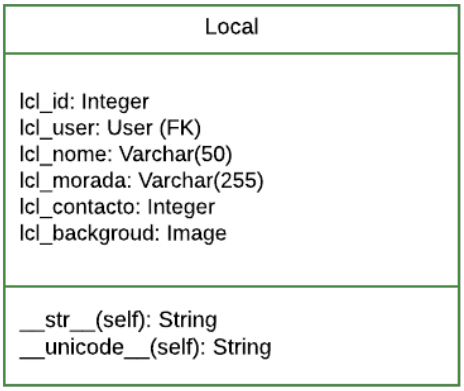


Figure 5.1: UML View of the class 'Local'

```
class Local(models.Model):

    class Meta:
        verbose_name_plural = "Local"

    lcl_id = models.AutoField(primary_key=True)
    lcl_user = models.ForeignKey(User, on_delete=models.CASCADE, verbose_name="Utilizador Associado")
    lcl_nome = models.CharField(max_length=50, verbose_name="Nome")
    lcl_morada = models.CharField(max_length=255, null=True, blank=True, verbose_name="Morada")
    lcl_contacto = models.IntegerField(null=True, blank=True, verbose_name="Contacto")
    lcL_background = models.ImageField(null=True, blank=True, verbose_name="Imagem de Fundo", upload_to='labqar')
```

Listing 5.1: Django model example

The following code shows the SQL Statement automatically created and used by the framework to create the table in the database.

```
CREATE TABLE labqar_local (
    lcl_id integer NOT NULL,
    lcl_nome character varying(50) NOT NULL,
    lcl_morada character varying(255),
    lcl_contacto integer,
    lcl_user_id integer NOT NULL,
    "lcL_background" character varying(100)
);

ALTER TABLE ONLY labqar_local
    ADD CONSTRAINT labqar_local_pkey PRIMARY KEY (lcl_id);

ALTER TABLE ONLY labqar_local
    ADD CONSTRAINT labqar_local_lcl_user_id_7c61fab1093443a4_fk_auth_user_id FOREIGN KEY (lcl_user_id) REFERENCES auth_user(id) DEFERRABLE INITIALLY DEFERRED;
```

Listing 5.2: SQL Statement for the model generation

By using the Django models, instead of creating and managing the entire database using SQL Statements, it is only required to define python classes; that would be necessary anyway

to build to web application itself. This method was used to replicate all of the database tables used by the legacy Atmis software.

5.1.1 The Connector Module

After setting up a new database similar to the one from the LabQAr, we implemented the data uploading tasks.

The Connector module makes the bridge between the two databases, and can be seen as if it was divided into two parts, one installed in the car and the other installed in the main server.

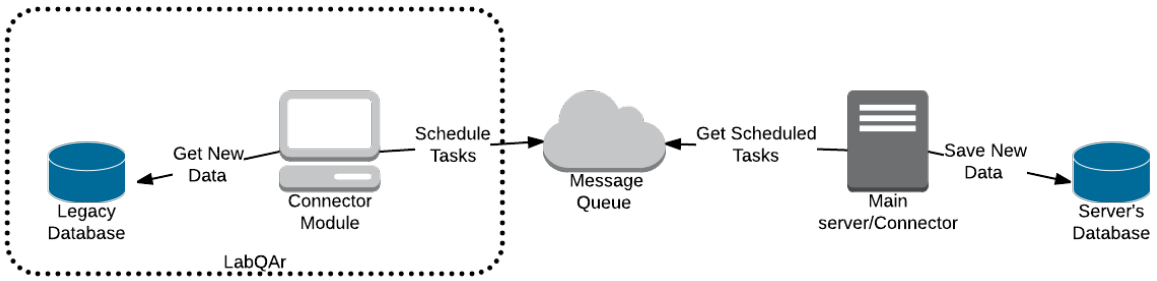


Figure 5.2: Connector Module Conceptual Behavior

The main part of the module is installed in the vehicle's desktop computer, and interacts directly with the local SQL Database using Python's pyodbc package, through common SQL queries. This module runs periodically, checking for updates, using a Windows shared task.

The code bellow shows the main part of the module. When it starts running, the module establishes a connection to the local database and tries to find a table named Connector_Helper, which is created by the module itself on the very first run. Connector_Helper acts as a log to control the incremental uploads. If the module is running for the first time in the computer or the database is new for some reason, it creates that table and inserts an entry for each database table whose entries are going to be exported to the main database. Each entry of the Connector_Helper table is composed by the name of the table to export and a number, which represents the ID of the last entry of that table that was already export.

Once the helper table exists, the Connector is ready to export the data. From that point on, it will start checking for new entries. Every method called since the helper table has been created is similar to the one demonstrated and explained bellow (differ only according to the table they work with). The method checks the ID of the last entry that was exported, from the helper tables, and compares it with the ID from the entries currently existing for that table. If the ID of an entry is bigger than the one saved on the helper table, it is scheduled as a Celery task, which will be explained shortly.

```
1 def adicionar_rede(cursor):
2     last_entry = cursor.execute("select ultimo_id from connector_helper where
    tabela_nome=\'Rede\'").fetchone()
3     last_entry_id = last_entry.ultimo_id
4
```

```
5     cursor.execute("select red_id, red_descricao from rede where red_id >
      last_entry_id")
6     for row in cursor.fetchall():
7         #celery task inserir_rede
8         inserir_rede.apply_async((row.red_id, row.red_descricao), retry=True,
      retry_policy={
9             'max_retries': 0,  # retry until sent
10            'interval_start': 60,  # start retrying one minute later
11            'interval_step': 30,  # wait more 30 seconds before each retry
12            'interval_max': 180, # wait at most 3 minutes for the next retry
13        })
14        last_entry_id = row.red_id
15
16    cursor.execute('UPDATE connector_helper SET ultimo_id=' + str(last_entry_id
      ) + ' WHERE tabela_nome=\'Rede\';')
17    cursor.commit()
```

Listing 5.3: Example of exporting new data entry from the table "Rede"

From the code above, it is important to notice the retry policy, which guarantees that, if a task fails it will be retried one minute later and then add 30 seconds to the waiting time and try again, until the task is assigned successfully. The waiting time may seem to be relatively high but this is because of the weak network connection inside the van.

The tasks are scheduled using the Celery package. A Celery task performs a dual role since it defines what happens when a task is scheduled (sent as a message), and what happens when a worker receives that message. Each task has a unique name, which represents a method in the web application code, that is referenced in the message so that the worker can find the right function to execute. In the Connector Module main part, the tasks are only used as an interface, to ensure the names are the same as the functions they will execute in the server and so do not take any action. This process can be compared with Remote Method Invocation, however it works asynchronously through the transmission of messages instead of direct method calls.

Configuring the program to use Celery implies some mandatory configurations:

- **Broker:** Celery needs a message broker to transport the messages. It offers support for both RabbitMQ and Redis servers, and some others in experimental phase. In this system it was chosen the RabbitMQ as the message broker but it can be easily changed if needed.
- **Serializer:** As well as most messaging services, the data needs to be serialized before being sent, which in Celery can be done using JSON, pickle, yaml and msgpack. In this case, we use JSON since it is simple to use and understand and is supported by most programming languages.
- **Additional Configurations:** Other configurations can be added but are not required. The set of additional configurations that where used can be seen in the excerpt bellow and are focused in reducing the number of unnecessary messages exchanged between the vehicle and the server to the minimum.

```
#connector.py
app = Celery('tasks')
app.conf.update(
    BROKER_URL='amqp://ijgtqkvn:RR3B2rRF-YQXBevJMxweXez-m61TBT_f@hare.rmq.cloudamqp.com/ijgtqkvn',
    CELERY_TASK_SERIALIZER='json',
    CELERY_ACCEPT_CONTENT=['json'],
    BROKER_POOL_LIMIT = 1, # Will decrease connection usage
    BROKER_HEARTBEAT = None, # We're using TCP keep-alive instead
    BROKER_CONNECTION_TIMEOUT = 30, # May require a long timeout due to Linux DNS timeouts etc
    CELERY_RESULT_BACKEND = None, # AMQP is not recommended as result backend as it creates thousands of queues
    CELERY_SEND_EVENTS = False, # Will not create celeryev.* queues
    CELERY_EVENT_QUEUE_EXPIRES = 60, # Will delete all celeryev. queues without consumers after 1 minute.
)

#tasks.py
@app.task
def inserir_rede(red_id, red_descricao):
    return
```

Listing 5.4: Celery configuration on the LabQAR and example declaration of a task

On the server side there is a complementary proccess. Here, the Connector does not have a standalone module but lives inside the Django application. While the celery configuration is equivalent, the big difference resides in the tasks declaration. In this case, the task is not a simple interface, but a method that receives the fields from the table entry, creates a new entry using the corresponding Django model and saves it to the live database.

```
#celery.py
app = Celery('tasks', broker='amqp://ijgtqkvn:RR3B2rRF-YQXBevJMxweXez-m61TBT_f@hare.rmq.cloudamqp.com/ijgtqkvn')

app.config_from_object('django.conf:settings')
app.autodiscover_tasks(lambda: settings.INSTALLED_APPS) #discovers celery settings inside django settings.py

#settings.py
CELERY_TASK_SERIALIZER = 'json'
CELERY_ACCEPT_CONTENT = ['json']
CELERY_IGNORE_RESULT = True
BROKER_POOL_LIMIT = 1 # Will decrease connection usage
BROKER_HEARTBEAT = None # We're using TCP keep-alive instead
BROKER_CONNECTION_TIMEOUT = 30 # May require a long timeout due to Linux DNS timeouts etc
CELERY_RESULT_BACKEND = None # AMQP is not recommended as result backend as it creates thousands of queues
CELERY_SEND_EVENTS = False # Will not create celeryev.* queues
CELERY_EVENT_QUEUE_EXPIRES = 60 # Will delete all celeryev. queues without consumers after 1 minute.

#tasks.py
@shared_task
def inserir_rede(red_id, red_descricao):
    novaRede = Rede(red_id, red_descricao)
```

```
22      novaRede.save()
```

Listing 5.5: Example of Celery configuration on the server and implementation of a task

To fetch the tasks from the message broker and complete the data flow, it is needed one last piece, the celery worker. The worker is a service that will check the broker for new tasks and execute the corresponding method in the web application. This is done by running the command bellow which is part of the script ran periodically by the CRON scheduler on the server.

```
1 celery worker --without-gossip --without-mingle --without-heartbeat --app=
    labqar -l info
```

Listing 5.6: Example of running a celery worker

5.1.2 Central Database

As the original database has more than 30 tables both Connector Module ends are ready to schedule and execute the tasks required to export all of them so the solution can be easily scaled in the future. This is also true for the server's database, which has all the Django models equivalent to all the tables. However, to reduce the number of unnecessary accesses to the original database from the Connector module in this project, all of the empty tables that are not used by the IDAD's team are excluded from the data export tasks.

Note that the solution aims to establish a connection between the data from the LabQAr and the final user. This is only possible by having a relationship between the users and the campaigns associated with them. Each user can have many campaigns and every campaign is associated to a specific location.

This logic was not available in the data from the original database so, apart from the tables already existing in the original database and that are exported to the server, in the server's database were created additional Django models that were required to implement this mapping between the data and the users. New entities were also added to support defining a maximum value to a given pollutant to setup up alerts of hazard situations and specific events.

Figure 5.3 illustrates the database created to serve the system. The models inside the red shape are the ones that were copied from the legacy database and that are required for the system, the others are the ones used for additional logic.

5.2 Integration API

Since the database and the web application are connected inside the server, the web services are only necessary for the Mobile Application.

The web services developed are RESTful web services, created inside the web application using the Django Rest Framework. They allow the mobile application to provide features and data similar to the one provided by the web application for external clients.

A REST API was setup in the web application by mapping an URL (in the urls.py file) to a method in the controller (views.py file) which will take care of the request and return

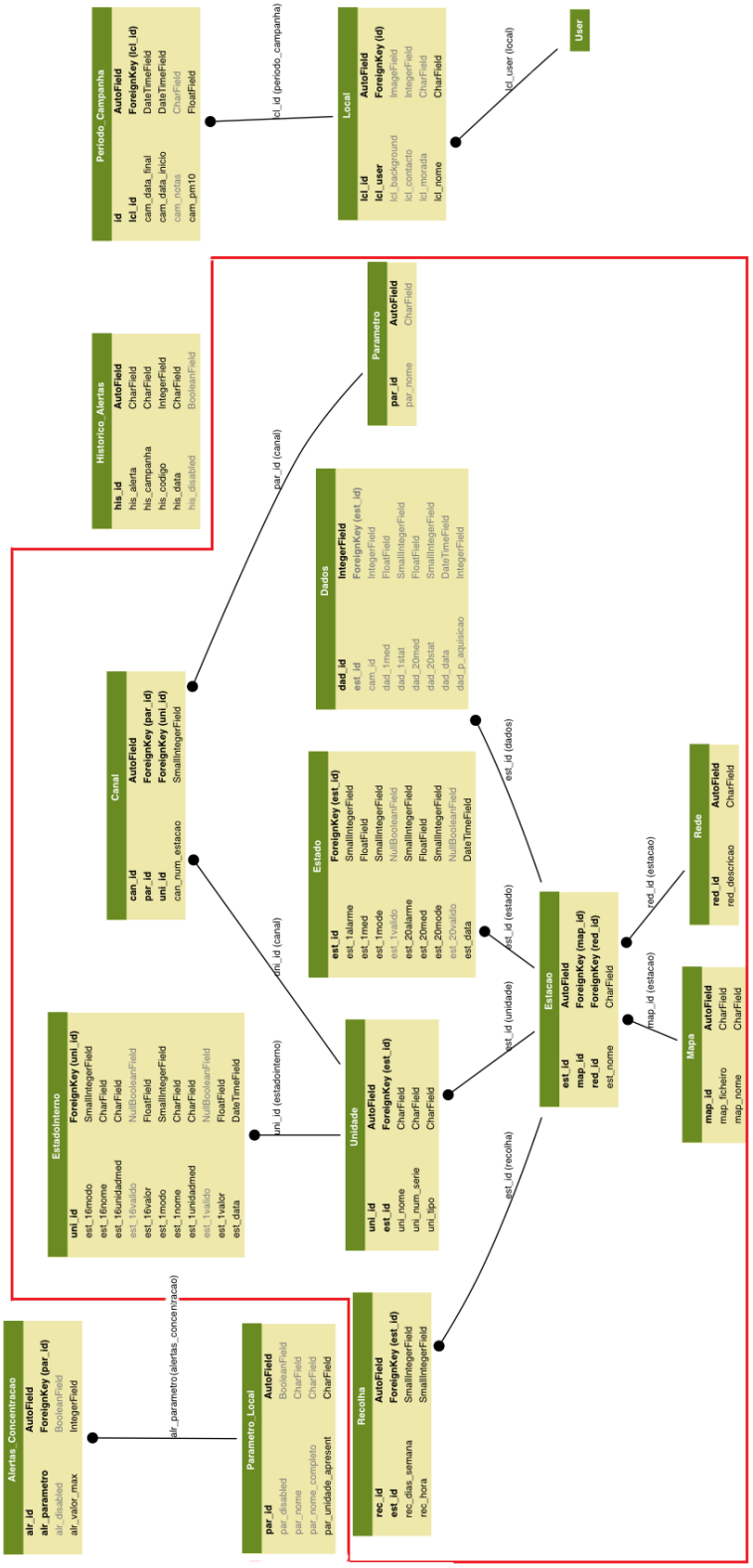


Figure 5.3: Central Database Diagram

the proper response containing the correct HTTP code and, if applicable, the requested data serialized as a JSON object.

```
#urls.py
url(r'^rest/latestupdate/(?P<id>[0-9]+)/$', views.get_latest_update),

#views.py
@api_view(['GET'])
@authentication_classes((TokenAuthentication,))
@permission_classes((IsAuthenticated,))
def get_latest_update(request, id):
    ...
    if last_update is not None:
        ...
        return Response(status=status.HTTP_200_OK,
                        data={'date_time': date_time, 'values':
serialized_values, 'campaign': serialized_campaign})
    else:
        return Response(status=status.HTTP_204_NO_CONTENT)
```

Listing 5.7: Example of setting up a RESTfull web service

From the code above, which demonstrates how to setup a RESTfull web service in the web application, it is important to highlight the decorators used for the method in the example. In this case, the API only accepts GET methods (@api_view['GET']). The other two decorators, are used to ensure that the API interface is only usable by users who have access to the system. This two decorators are used for every method from the API interface, except for the login.

The table 5.1 presents the requests available from the REST API and the corresponding responses. First, when a user successfully logs into the system (request 1) from his mobile phone, through the rest API, he receives, apart from other details, a Token that is needed to authenticate every subsequent call to the REST API. The login is the only operation that does not require the token, since it already forces the user to have a valid account by requiring his user name and password.

Request 8 is normally used after log in. In this request, the id of the user is passed in the POST, along with a token. This token does not have anything to do with the authentication token but it is needed for the push notifications system. Inversely, request number 2, the log out operation, is simply used to inform the server that the user logged out and, therefore, should not receive more push notifications.

For request number 7, in the POST request are passed the details that the user wants to edit and in the response it gets the same details with the information if they were updated or not.

All the other requests are related to data access and are pretty much self explanatory.

#	REQUEST	TYPE	RESPONSE	PURPOSE
1	rest/login/username /password/	GET	{ 'username': username, 'name': first_name, 'superuser': True/False, 'id': user.id, 'token': token }	Authenticates a mobile user into the system.
2	rest/logout/	POST	Status code only, no data	Logs an user out of the system and disables push notifications.
3	rest/latestupdate /userID/	GET	{ 'date_time': date_time, 'values': values, 'campaign': campaign }	Obtains the latest values available for the specified user.
4	rest/latestupdate /userID/airindex/	GET	{ 'date_time': date_time, 'index': index, 'campaign': campaign }	Obtains the latest air quality classification for the specified user.
5	rest/latestupdate /userID/chart /airindex/	GET	{ 'index_chart': index_chart }	Obtains the air quality classification for the specified user during the last 24 hours.
6	rest/latestupdate /userID/chart /pollutant/	GET	{ 'value': values }	Obtains the variation of a selected parameter during the last 24 hours available to the specified user.
7	rest/editsettings/	POST	{ 'name_changed': T/F, 'name': user.first_name, 'username_changed':T/F, 'username': username, 'password_changed':T/F }	Updates the user personal details with the given values.
8	rest/savetoken/	POST	Status code only, no data	Refreshes the GCM token for the user's device and enables push notifications.
9	rest/alerts/userID	GET	{ 'alerts': alerts }	Gets a list of the most recent notifications received by the user.

Table 5.1: REST interface requests, type and sample responses

5.3 Website

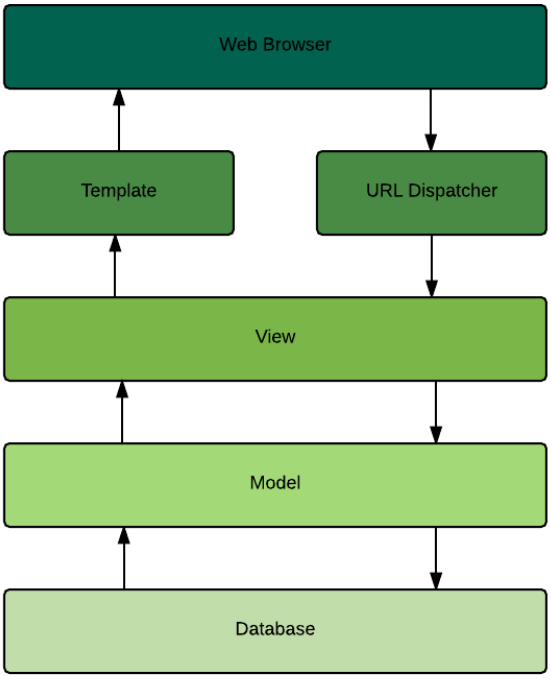


Figure 5.4: Django Overall Project Structure

5.3.1 Structure Overview

The Django framework functional structure, Figure 5.4, follows the Model-View-Template (MVT) pattern [23] which plays a central role in every Django application. This pattern can also be analogously compared to a Model-View-Controller (MVC) pattern [24], which might be more familiar since it is extremely popular for building web applications.

In the MVC pattern there are, as the names suggests, three components: the model, the view and the controller. The model represents the data, the logic and rules of the application; the view is related to the representation of the data and the controller is responsible of accepting some sort of input and convert it in commands for the model or the view. [25]

Comparing to the MVT pattern, it is visible that all the three components are there although they don't use the conventional names:

- **Model:** A model in Django represents the data and each model represents a database table.
- **View:** The view in the MVC is equivalent to the Template in the MVT. It is where the data is presented, in this case by the generation of HTML code.
- **Controller:** The controller component in the MVC corresponds to the View in the MVT pattern plus the Django URL Dispatcher. This two pieces (View and URL Dispatcher), when put together, are capable of receiving an input, in the form of an URL

and map it to a view, which can use the Model component to retrive/store data for the Template (View in MVC) component to present it.

5.3.2 Technology Stack

Building a web application requires inevitably using a lot of different technologies since it requires working all the way up the stack, from the database to the interface. Figure 5.5 shows all the technologies that were used to build the web application for this system.

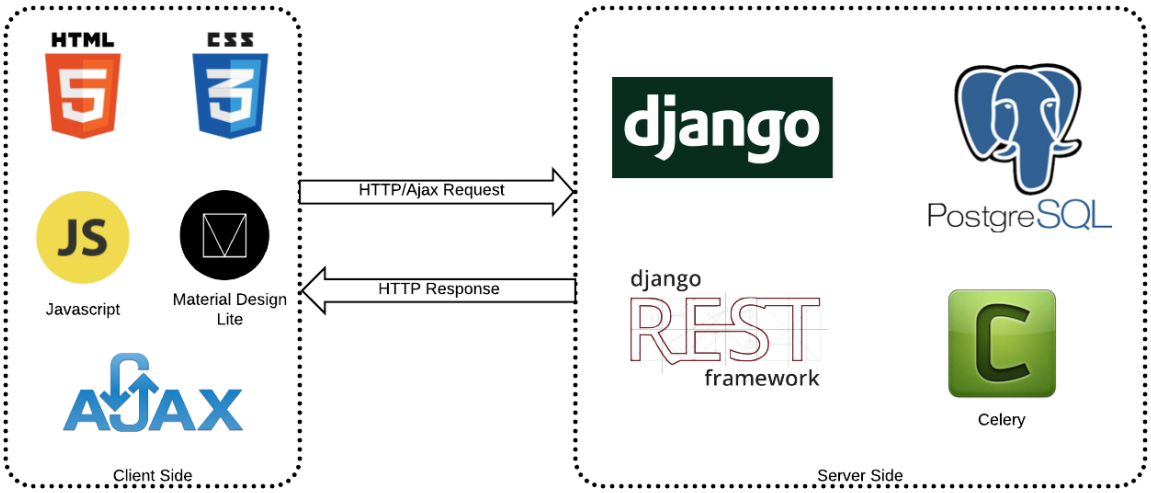


Figure 5.5: Stack of Tecnhologies used in the website

- **HTML5+CSS3+Javascript:** HTML is the standard markup language for working with web pages. Together with CSS and Javascript, they are the cornerstone technologies used to create web pages, as well as user interfaces for mobile and web applications.
- **Material Design Lite:** Material Design Lite (MDL) is a Google's framework that allows the Material Design experience using vanilla Web technologies like HTML, CSS and Javascript. Material Design is the Google's guidelines for building Android applications, so, the choice of the MDL framework falls upon the interest of maintaining a consistent look and feel between the Web and the Mobile applications. It is similar to the Bootstrap framework but it offers a "Material" implementation of the components.
- **Ajax:**Ajax is not viewed as a technology itself, but a group of technologies which allows for web pages to get and update content dynamically, without the need to reload the entire page.
- **Django Rest Framework:** Django REST framework is a powerful and flexible toolkit for building RESTfull APIs. The use of this technology in the system will be explained more in depth in the next section.

5.3.3 Supported Interactions

The Web Application provides a set of features more comprehensive than the mobile application. The set of features required are detailed in the Requirements Chapter and in this

section it is going to be given an explanation of the implementation of each main features and activities.

All the data presented in the screenshots of the application is demonstrative and does not correspond to identified campaigns.

The structure of the application follows the natural structure of a Django Application (Figure 5.6), highlighting the models, views, templates and urls.

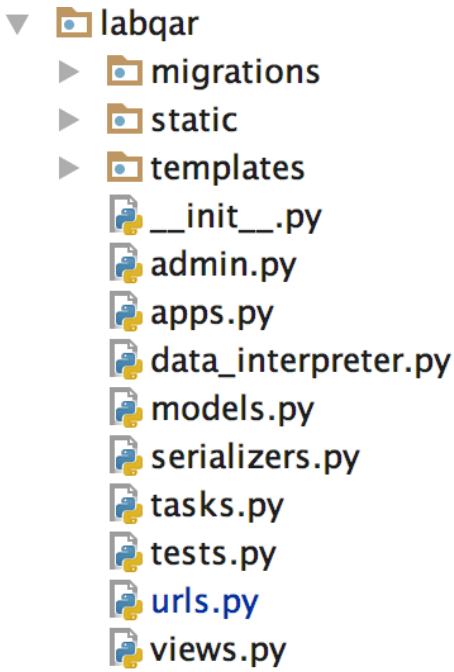


Figure 5.6: LabQAr Django App Structure

Before analysing the implemented features, some components of the implementation require a brief explanation. Most of the interface was developed using a combination of plain HTML5+CSS3 and the HTML components from the MDL framework, however this solution does not offer direct support to the presentation of charts so it was necessary to resort to an external Javascript library. The library chosen was the Google Charts library because it offers cross-browser support and has a look that fits into the Material Design concept.

To display the Google Chart, first it is added an empty div to the page and then it is used Javascript to draw it programmatically when desired. In the line 10 of the code bellow, the data is dynamically added to the chart from a template variable that was passed to the context of the HTTP response when the page loaded or obtained from an ajax request, when applicable. On line 17 the chart is created, associated to the empty div and it is drawn in the next line. The rest of the code, from line 20 and forward is necessary to make the chart responsive with the rest of the interface, which does not happen otherwise.

```
1 <script type="text/javascript">
2     google.charts.setOnLoadCallback(drawChart);
3
```

```
    function drawChart() {
    // Create the data table.
    var data = new google.visualization.DataTable();
    data.addColumn('string', 'Valor');
    data.addColumn('number', ' ndice');
    data.addColumn({type: 'string', role: 'style'});
    data.addRows({{ index_values | safe}});

    // Set chart options
      var options = {
          // Names, colors, animations, etc
      };

    var chart = new google.visualization.ComboChart(document.getElementById('
    chart_div'));
    chart.draw(data, options);

      function resizeHandler() {
            chart.draw(data, options);
    }

    if (window.addEventListener) {
        window.addEventListener('resize', resizeHandler, false);
    }
    else if (window.attachEvent) {
        window.attachEvent('onresize', resizeHandler);
    }
    }
    </script>
```

Listing 5.8: Example of creating a Google Chart to embed in a Web page

Ajax is also used in the client side of the application. It is only used when a user chooses to view a 24h chart for a given parameter or of the variation of the quality index. This data is not required until the user specifically asks for it and only the data from one parameter is required at a given time. For this reason, using an Ajax call is the best way to load the data on demand instead of heaping it on a page's template variable.

The last different component is a JQuery Datepicker. It is the datepicker that is used in the website to select specific days. The picker is declared inside a HTML form element. When a date is selected from the picker, it is passed to the application through the form. Before the picker displays each month, it checks if any of days of that month is unavailable for the logged user.

5.3.3.1 Push Notifications

Even though push notifications do not interact directly with the website, they still need to be integrated in the web application. The notification support is provided by Django Push Notifications App, that is installed in the server along with the main application. This app implements two additional models, one to store Android devices information (GCMDevice) and other to store iOS devices information (APNSDevice). The second model is not needed since the system does not use an iOS application at this stage.

When the user logs in successfully from the mobile application, the server receives his user ID and the GCM token for his device through a REST request. This process is explained in more detail in the Mobile Application section. It then checks whether the user already has that device registered to receive notifications and, if not, registers the new device.

```
def save_gcm_token(request):
    id = request.POST.get('id')
    try:
        user = User.objects.get(pk=id)
        registration_id = request.POST.get('registration_id')

        if user is not None and registration_id is not None:
            devices = GCMDevice.objects.filter(user=user)

            if not devices:
                device = GCMDevice(user=user, registration_id=registration_id)
                device.save()
            else:
                new_device = False
                for device in devices:
                    if device.registration_id != registration_id:
                        new_device = True
                    else:
                        device.active = True
                        device.save()

                if new_device:
                    device = GCMDevice(user=user, registration_id=
    registration_id)
                    device.save()

            return Response(status=status.HTTP_200_OK)
        else:
            return Response(status=status.HTTP_400_BAD_REQUEST)
    except:
        return Response(status=status.HTTP_400_BAD_REQUEST)
```

Listing 5.9: Registering a GCMDevice

A Push notification is sent when a staff member has previously set a limit value for a parameter, through the Django Administration interface and the data model receives a new entry where the value for the correspondent parameter is above the limit specified. The notification is sent to a user if the entry belongs to a campaign associated with him and to every staff member, as long as they are logged in the mobile application.

To make this validation a custom method is required to validate the data saved to the model and relate it to the right model with the post_save.connect() method.

```
def on_data_save(sender, instance, **kwargs):
    if kwargs['created']:
        # Logic to select the devices to receive the notification
        ...

        devices = GCMDevice.objects.filter(user__is_staff=True)
        devices.send_message(Alerts.getAlertMessage(par.par_nome_completo)

```

```
9 post_save.connect(on_data_save, sender=Dados)
```

Listing 5.10: Defining a custom validation when saving a Data

5.3.3.2 Session Login

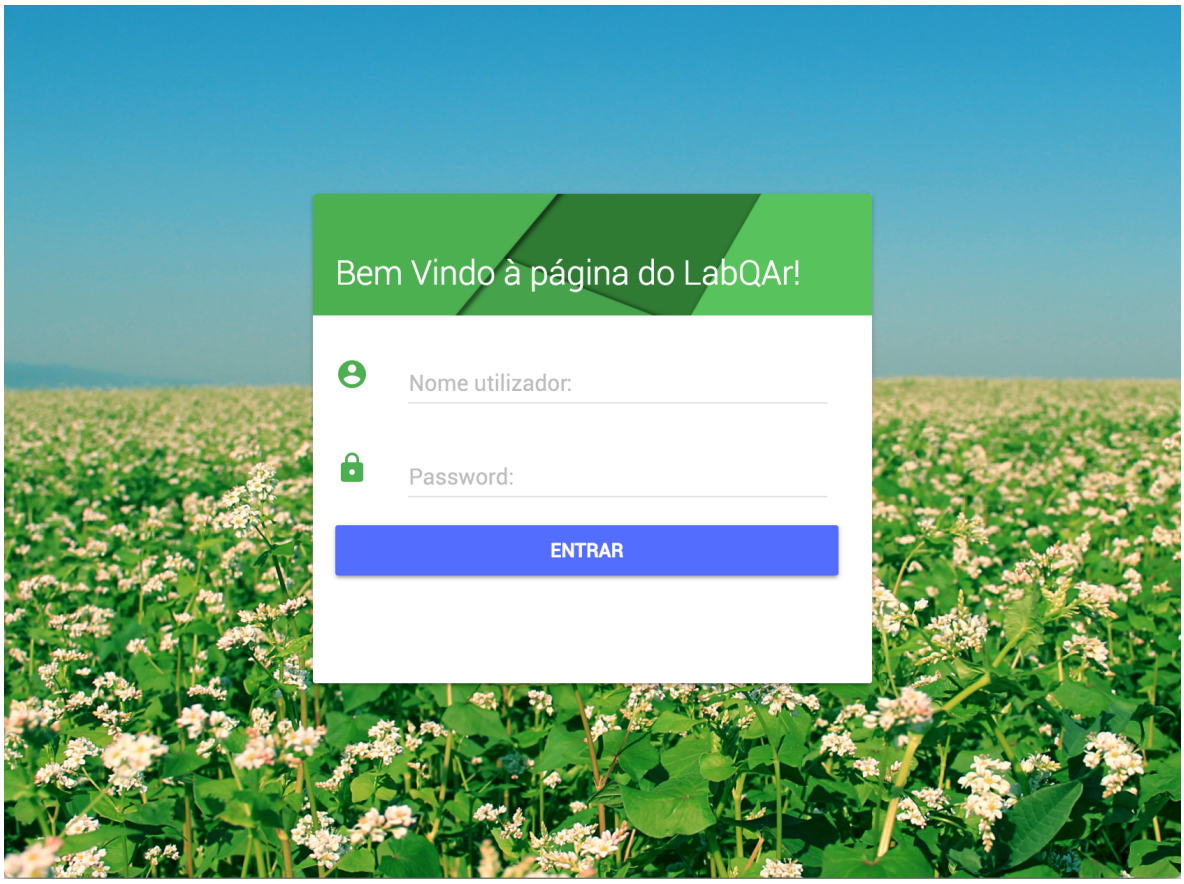


Figure 5.7: Login page

When the user accesses the main web page, if he is not logged in yet, he is redirected to the log in page, Figure 5.7, where he gets an welcome screen to input his log in details. When the log in button is pressed, the application tries to validate the user through the Django authentication system. If the data is invalid, the pages shows a error message, otherwise it forwards the user to the main page.

After the login, the background can be different (defined by a staff member) according to the logged user/organization, allowing for a more customized experience.

5.3.3.3 Overview Of The Air Quality

Figure 5.8: Overview page

The Figure 5.8 shows the general overview of air quality classification in the selected moment as the classification of each pollutant involved in the calculation. To display the bar chart, we use the Google Charts library; the library does not support the use of half pie charts so, in this case it is used an SVG Path to draw the arch.

The first card displays a general classification of the air quality. This result is given by classification of the worst parameter, which are all visible in the chart. Each parameter is classified with a different color and level, according to the hazard level associated with its concentration. The half pie charts also displays the corresponding color and displays a bigger 'slice' the higher the hazard level is.

5.3.3.4 Overview Of The Air Quality In The Previous 24H

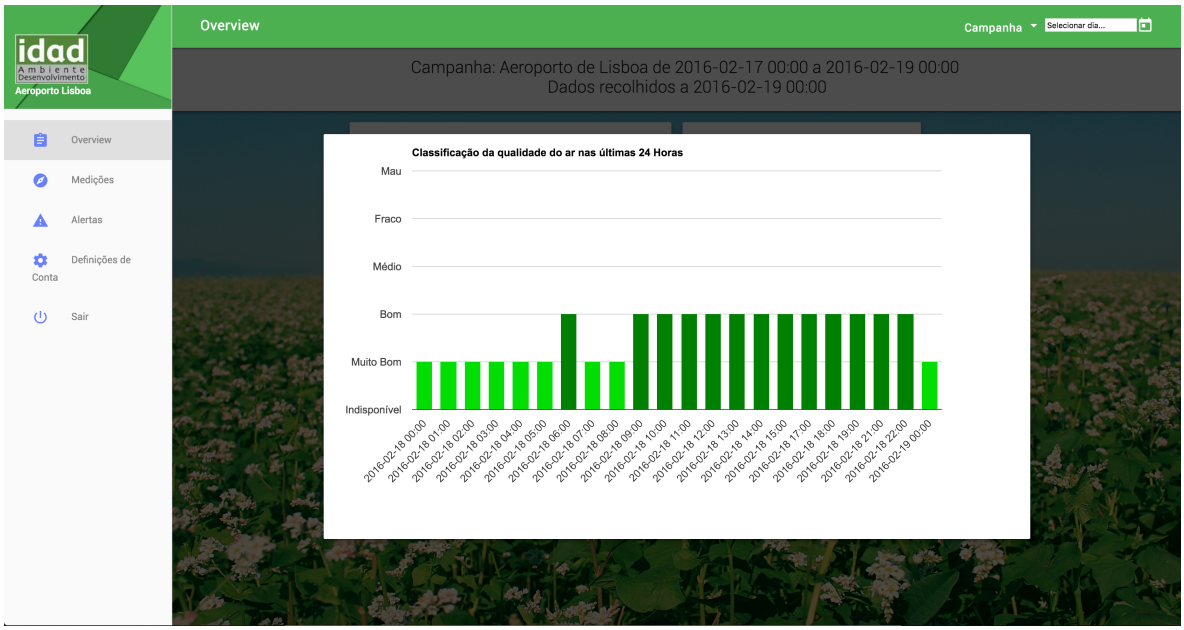


Figure 5.9: Overview the previous 24H overlay

From the main page, Figure 5.9, if the user presses the button marked with the number one, he is presented with a chart overlaying the content of the pages that shows the variation of the air quality classification over the past 24 hours.

This is done through an hidden div that is only made visible when the user presses the said button. The div is then used to draw the chart with the Google Charts library.

5.3.3.5 Select a Campaign or Day

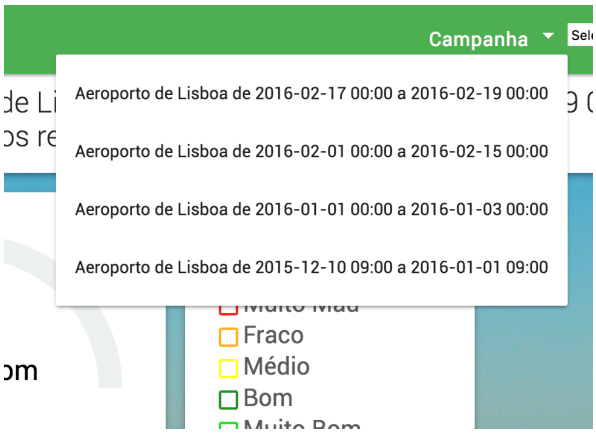


Figure 5.10: From a Dropdown List.

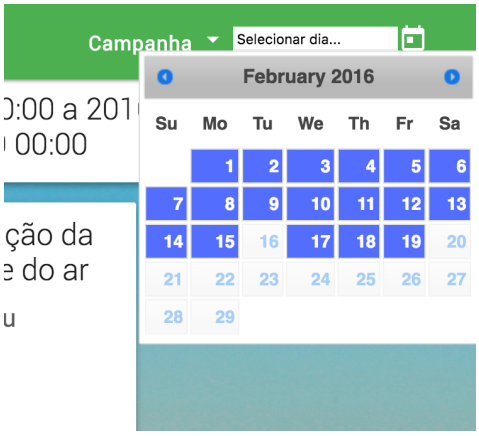


Figure 5.11: From the calendar.

Figure 5.12: Diferent ways of selecting campaign

From the main page the user can change the campaign (or the day) that he is viewing (Figure 5.12.

This can be done either by selecting a campaign from the dropdown menu , Figure 5.10, or selecting a day from the calendar picker, Figure 5.11. In the calendar, the campaigns corresponding to the logged user are marked with a blue color and the days that are not available to them are disabled in the calendar. This makes the choice of a day intuitive for the user.

These actions correspond to the buttons marked with number 2 and 3, respectively, in the Figure 5.8.

5.3.3.6 Change user details

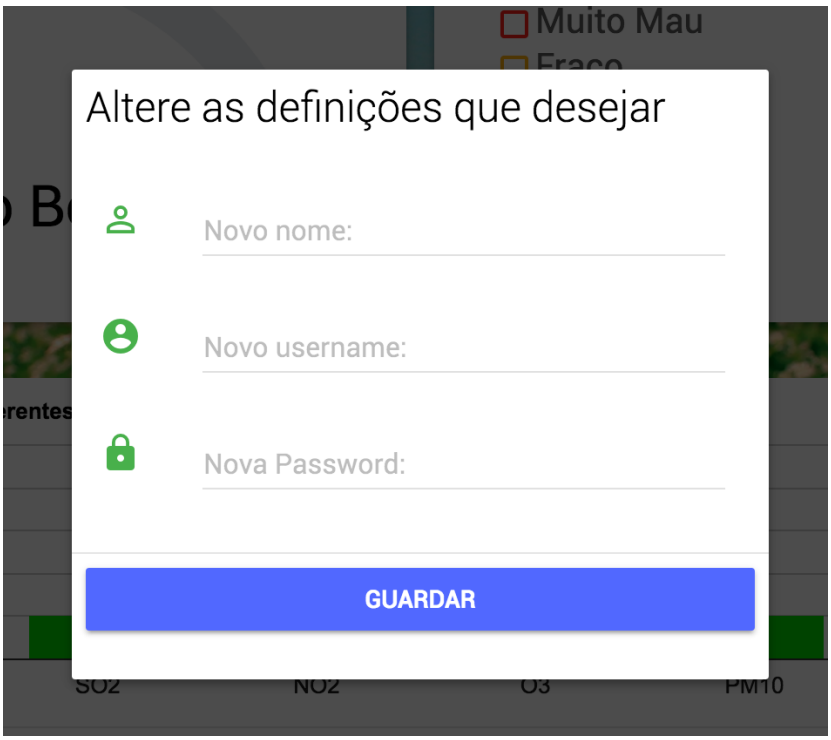


Figure 5.13: Change Details Overlay

If the user wants to change his details, he selects the button number five in the Figure 5.8. He is presented with an overlapping div, which starts hidden, with the details that he is allowed to change. The user can change a single field at a time, two fields at a time, or all the fields at the same time. In case the operation succeeds, the data is updated, otherwise it is shown an error message for each field that the user tried to update unsuccessfully.

5.3.3.7 Measurements Of The Different Parameters

Figure 5.14: Measurements Page

The button number four from Figure 5.8 takes the user to a page (Figure 5.14) where he can check the values for all the different parameters. This page, besides maintaining available most of the features described before, also unlocks new features. From here, the user can visualize the charts for both the variation of a parameter during the previous 24 hours or for each day in the duration of the campaign. To do this, it is only required to select a parameter which will display an invisible div, just like the others chart. From here the user can choose the 24 hours view (Figure 5.15) or the campaign (Figure 5.16), case that is only available if the selected day belongs to a campaign.

Figure 5.15: Measurements 24 Hour Variation Overlay

Figure 5.16: Measurements Campaign Variation Overlay

5.3.3.8 Check alerts history

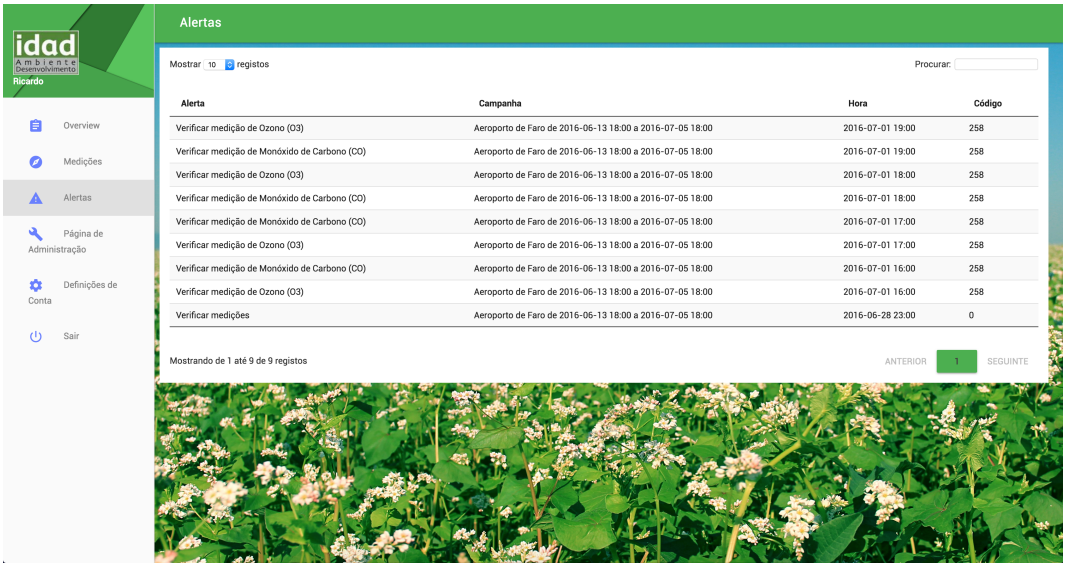


Figure 5.17: Alerts History Page

The button number seven from Figure 5.8 takes the user to a page, as seen on Figure 5.17, where he can check all the data driven alerts that where detected. These alerts include both hazard level alerts, alerts related to sensors' malfunction errors and long periods of time without receiving data.

On this page, the user can search alerts by their name, associated campaign, date and error code.

5.3.4 The Administrator interface

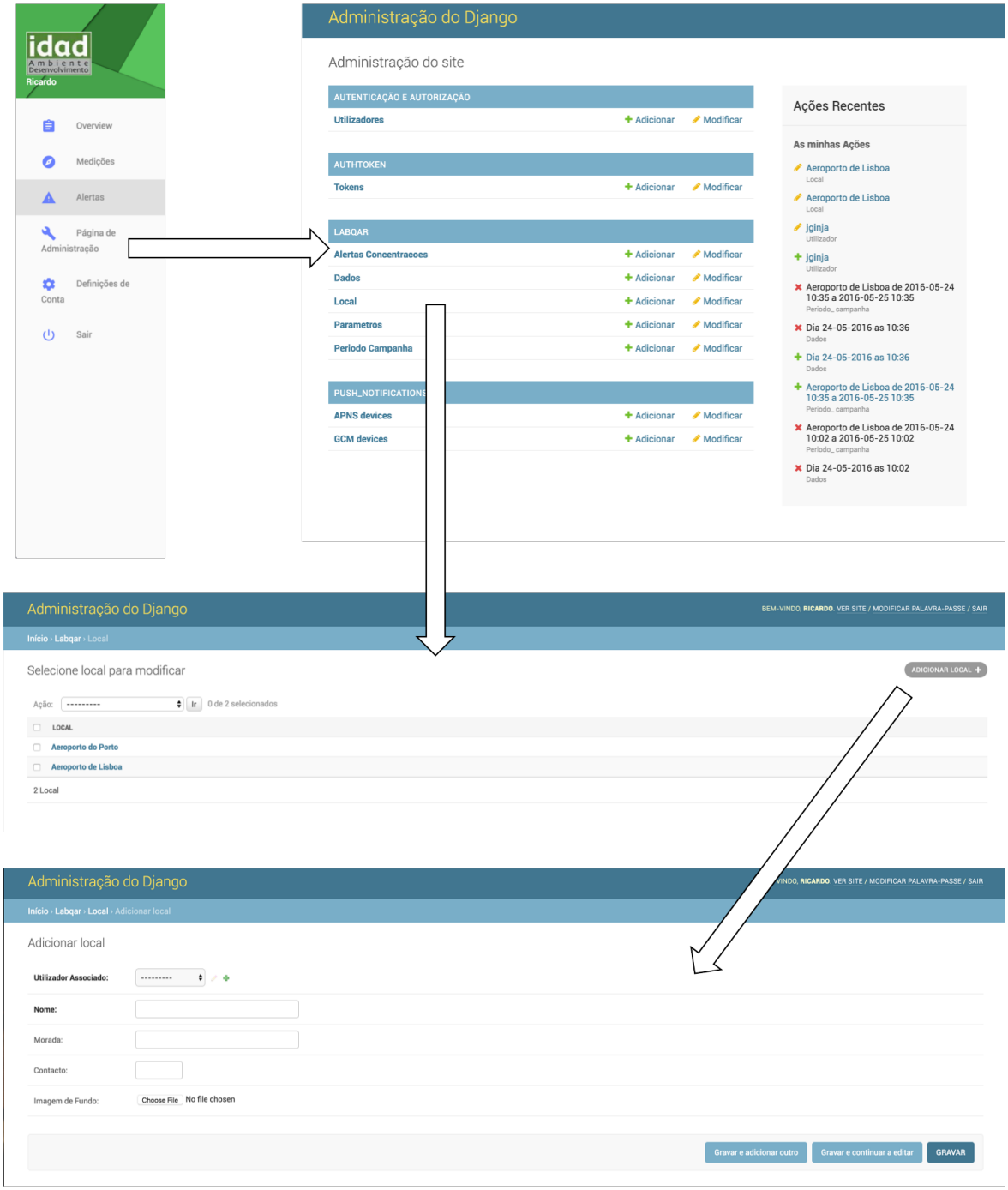


Figure 5.18: LabQAr's Administrator interface

For the Staff users, an additional option is available from the drawer menu, the Administrator Page. The administrator page (Figure 5.18) is generate automatically by the Django Framework from the models used in the application.

The Django Administrator provides an interface to visualize, edit, remove or add new entry through a graphical form originated by the Django Framework and based on each model fields.

For the model Periodo_Campanha it is necessary to guarantee that the end date is always bigger than the start date, which is done by implementing a custom form for that model in the admin.py file.

```
class PeriodoForm(ModelForm):
    class Meta:
        model = Periodo_Campanha
        fields = ['lcl_id', 'cam_data_inicio', 'cam_data_final', 'cam_notas', '
    cam_pm10']

    def clean(self):
        start_date = self.cleaned_data.get('cam_data_inicio')
        end_date = self.cleaned_data.get('cam_data_final')
        if start_date > end_date:
                raise forms.ValidationError("A data de inicio nao pode ser
    superior a de fim.")
        return self.cleaned_data


class PeriodoAdmin(admin.ModelAdmin):
    form = PeriodoForm
```

Listing 5.11: A custom form sample used to extend Django's automatic model-view generation.

5.4 Mobile Application

The Mobile Application intends to provide the simplest and fastest way to access the most recent data that each user has access to, so therefore it does not require to implement all the functionalities implemented by the web application. The set of features required are detailed in the Requirements Chapter and in this section it is going to be given an explanation of the implementation of each main features and activities.

All the data presented in the screenshots of the application is merely demonstrative and does not correspond to real data.

5.4.1 Overview

Before getting to the implementation of each feature, it is important to give some general considerations.

The mobile application is built to run in any device using Android 4.0 or any newer version, which at the time represents 97.4% of the Android devices. The design chosen for the application follows the guidelines of the Google's Material Design, making it have a consistent "look and feel" with other Android applications and the Android operative system itself.

To make it more pleasant to user, the application also offers a set of animations so that it does not look like simply some charts on a screen but a comprehensive application.

Apart from the core dependencies, some aditional dependencies were added to the project.

```
dependencies {
    compile fileTree(dir: 'libs', include: ['*.jar'])
    testCompile 'junit:junit:4.12'
    compile 'com.android.support:appcompat-v7:23.3.0'
    compile 'com.android.support:design:23.3.0'
    compile 'com.android.support:cardview-v7:23.3.0'
    compile 'com.squareup.retrofit2:retrofit:2.0.2'
    compile 'com.squareup.retrofit2:converter-gson:2.0.1'
    compile 'com.squareup.okhttp3:okhttp:3.2.0'
    compile 'com.github.PhilJay:MPAndroidChart:v2.2.4'
    compile 'com.google.android.gms:play-services:8.4.0'
}
```

Listing 5.12: Aditional libraries

- **com.android.support:** All the dependencies added from com.android.support provide all the components and features required to implement the Material Design components and also offer the backwards compatibility of those components to older Android versions
- **com.squareup:** All the dependencies from this repository are used to perform the requests to the REST API and obtaining the respective response.
- **com.github.PhilJay:MPAndroidChart:v2.2.4:** MPAndroidChart was the library used to create and animate all the charts used in the application
- **com.google.android.gms:play-services:8.4.0:** Offers the access to the Google Play Services, which in this case is required for the Google Cloud Messaging Services.

Interacting with the website data requires the use of the REST API provided by the same, which was done with the help of the Retrofit library. Retrofit is a type-safe HTTP client for Android and Java. The first step to setup Retrofit [1] is to turn the REST API into a Java Interface. The following example shows how this is done for one request. In this example, it is important to notice the @Header("Authorization") attribute where it is passed the authentication token that is required by the API, as it was mentioned in the section before.

```
public interface ApiInterface {
        @GET("latestupdate/{id}")
        Call<Measurement> getLatestUpdate(@Header("Authorization") String
    authorization, @Path("id") int id);
    }
```

Listing 5.13: Example of turning a REST API Request into a Java Interface

After defining the Java interface, a service is created that uses that interface to validate the HTTP requests. This was done using the singleton pattern [26], because the same instance

[1] Retrofit - http://square.github.io/retrofit/

can be used to process the requests across the different segments of the application. In the code sample bellow, the first time the getService() method is called, creates a retrofit instance which uses the base url for the API and GSON [2] to parse the JSON responses into Java objects and the creates the Api service using the Java interface. Every other call to the getService() method will simply return the same instance of the service.

```
private static final String API_BASE_URL = "http://idad-qualar.rhcloud.com/rest
    /";

private static Retrofit retrofit = null;

private static ApiInterface apiService = null;

public static ApiInterface getService() {
    if(retrofit == null){
        retrofit = new Retrofit.Builder()
                .baseUrl(API_BASE_URL)
                .addConverterFactory(GsonConverterFactory.create())
                .build();
        apiService = retrofit.create(ApiInterface.class);
    }
    return apiService;
}
```

Listing 5.14: Example of declaring a Retrofit instance

Before being able to perform requests to the server, we have to convert the JSON responses into Java objects. This is done with the help of the GSON library, which converts objects in both ways, back and forth. The interface created in the first example defines the class Measurement in the HTTP request (Call<Measurement>), which tell the request which class should be used to parse the response into a Java Object. This class is a simple Java class where the attributes are mapped to the corresponding JSON attributes by the usage of the tag @SerializedName(attribute_name).

As an example, the following JSON response:

```
{
    'datetime':  "2016-07-01 09:00",
    'values':  {
                "PM10": {
                    "value": 15.0,
                    "stat": 0,
                    "name": "Particulas (PM10)",
                    "unit": "  g/m3"
                },
                ...
                "O3": {
                    "value": 47.69,
                    "stat": 0,
                    "name": "Ozono (O3)",
                    "unit": "  g/m3"
                }
            },
    'campaign':  "Campanha de 2016-02-17 00:00 a 2016-02-19 00:00"
}
```

[2] GSON - https://github.com/google/gson

is converted into a Java Object by the following class:

```
public class Measurement {

    @SerializedName("date_time")
    private String dateTime;

    @SerializedName("values")
    private HashMap<String,Pollutant> values = new HashMap<>();

    @SerializedName("campaign")
    private String campaign;

    /* Getter and Setter methods */
}
```

Listing 5.15: Example of class that converts a JSON object to a Java object

Another important part of the development of this application was the integration of the charts. Android does not offer native support for the implementation of charts so it was used an external library named MPAndroidChart. The library supports the integration of charts as normal Android components, which makes them easier to integrate. The following excerpt shows how to integrate a Bar Chart in the layout of an activity:

```
<android.support.v4.widget.SwipeRefreshLayout>
    ...
    <com.github.mikephil.charting.charts.BarChart
        android:id="@+id/barChart"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:layout_weight="1" />
    ....
</android.support.v4.widget.SwipeRefreshLayout>
```

Listing 5.16: Example of adding a chart in the layout of an activity

Afterwards the chart can be fully customized with the definition of colors, animations, zooming, etc. The data is added to the chart programatically, after being obtained from the server through the REST API, using the process that was explained above. In this case, the chart uses vertical bars so the values corresponding to that axis are converted into a BarDataSet which creates the bars with the right values and colors which then are mapped to the correspondent label of the horizontal axis. After the BarData is added to the chart, the invalidate() method is essential to force the chart to redraw.

```
BarDataSet set1;
ArrayList<BarEntry> yVals = new ArrayList<>();
ArrayList<String> xVals = new ArrayList<>();

int count = 0;
for (Object value : values) {
    ArrayList<String> value_array = (ArrayList<String>) value;
    yVals.add(new BarEntry(Float.parseFloat(value_array.get(1)), count));
    xVals.add(value_array.get(0));
    count++;
}
set1 = new BarDataSet(yVals, "");
set1.setColors(colors);
set1.setDrawValues(false);
```

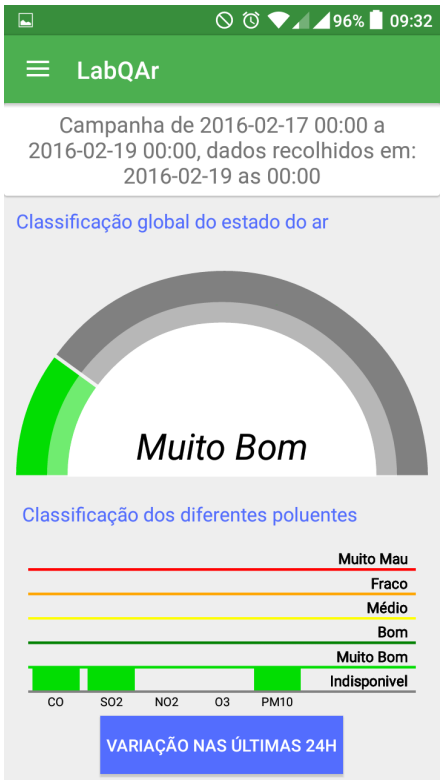


Figure 5.19: Overview screen

```
15
16 ArrayList<IBarDataSet> dataSets = new ArrayList<>();
17 dataSets.add(set1);
18 BarData data = new BarData(xVals, dataSets);
19
20 mBarChart.setData(data);
21 mBarChart.invalidate();
```

Listing 5.17: Adding data to the chart

During the development of the mobile application, it was discovered that neither this library or any other charts library offered native support to half pie charts, as the one used in the Overview screen, as seen in the figure 5.19. Since this was the chart of choice to maintain the chart consistent with the usual visualizations in this domain, we decided to use the full pie chart offered by the library used. But to achieve the half pie circle, the chart is divided in half by an invisible View that is used as a guide for the start of a LinearLayout that paints the rest of the screen with the background color, to hide the second half of the pie chart.

```
1 <RelativeLayout
2     android:id="@+id/pieChart_layout"
3     android:layout_width="match_parent"
4     android:layout_height="match_parent"
5     ...
6     >
7
8     <com.github.mikephil.charting.charts.PieChart
9         android:id="@+id/pieChart"
```

```
        android:layout_width="match_parent"
        android:layout_height="match_parent" />

    <View
        android:id="@+id/empty_view"
        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:layout_centerInParent="true"></View>

    <LinearLayout
        android:id="@+id/barChart_layout"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:layout_below="@+id/empty_view"
        android:background="#EEEEEE"
        android:orientation="vertical">

        ...
    </LinearLayout>
</RelativeLayout>
```

Listing 5.18: Transforming the PieChart into a HalfPiechart

The last setup that should be highlighted is the push notifications service. The application is ready to receive push notification by the use of the Google Cloud Messaging service. To prepare the application to receive notification are declared one receiver and three services in the AndroidManifest.xml file, as shown in the next excerpt.

```
<receiver
    android:name="com.google.android.gms.gcm.GcmReceiver"
    android:exported="true"
    android:permission="com.google.android.c2dm.permission.SEND" >
    <intent-filter>
        <action android:name="com.google.android.c2dm.intent.RECEIVE" />
        <category android:name="com.ua.ricardomartins.qualar" />
    </intent-filter>
</receiver>

<service
    android:name=".MyGcmListenerService"
    android:exported="false" >
    <intent-filter>
        <action android:name="com.google.android.c2dm.intent.RECEIVE" />
    </intent-filter>
</service>

<service
    android:name=".MyInstanceIDListenerService"
    android:exported="false">
    <intent-filter>
        <action android:name="com.google.android.gms.iid.InstanceID"/>
    </intent-filter>
</service>

<service
    android:name=".RegistrationIntentService"
    android:exported="false">
</service>
```

```
<service
    android:name=".UnregisterIntentService"
    android:exported="false">
</service>
```

Listing 5.19: Adding GCM in the Android Manifest

The receiver extends a WakefulBroadcastReceiver that receives a device wakeup event and then passes the work off to a Service, while ensuring that the device does not go back to sleep during the transition. It receives GCM messages and delivers them to an application-specific GcmListenerService subclass that should be declared in the AndroidManifest.xml file. This subclass receives the message sent from the GCM server to the service and turns it into the notification to push.

Another required service is the InstanceIDListenerService. This service is activated when the GCM token changes, and notifies this change to a subclass of InstanceIDListenerService that is also mapped to the service in the manifest. The role of this subclass is to force the execution of the RegistrationIntentService.

The RegistrationIntentService is an IntentService which is handled as an asynchronous request that runs on a separate thread and stops itself when it runs out of work. The purpose of this service is to register the device in the App's GCM Server and to obtain its Registration Token. This service is executed every time the main activity is loaded and, because of the previous service, every time the GCM token is changed for some reason. Once the device is registered in the GCM Server and the token is received, it is manually sent (through a REST service) to the main server, ensuring that the user always receives his notifications as long as he is logged in.

As a final note, it is also important to mention that the application always shows the most recent data, as long as a data connection is available. If the user leaves the application on background, when he returns the application automatically updates the current view with new data, if available.

5.4.2 Supported Interactions

5.4.2.1 Session Login

When the user starts the application, it will first check if the user has already logged into the application. This is done by storing a boolean as in the Android Shared Preferences every time a user logs into the application and changing it to false at the moment of the log out.

Before the application requests the log in to the server, after the user press the button, it checks if the user inserted both the username and the password. If any or both are missing, it tells the user through the display of an error next to the missing field. In case the user filled both field, the request is sent to the server.

From the moment the request is sent, there are three possibilities. If the phone has no Internet connectivity, a Snackbar is displayed that informs that the operation was not possible and the user should try again. On the other hand, if the connection is available but the server

fails to authenticate the user, it is displayed a message informing that the data is incorrect. Last, if everything is alright, the user is logged into the application and is presented with the home activity.

The Figure 5.20 shows the possibilities for the login feature on the application itself, the Figure 5.21 shows the activity diagram for the same feature.

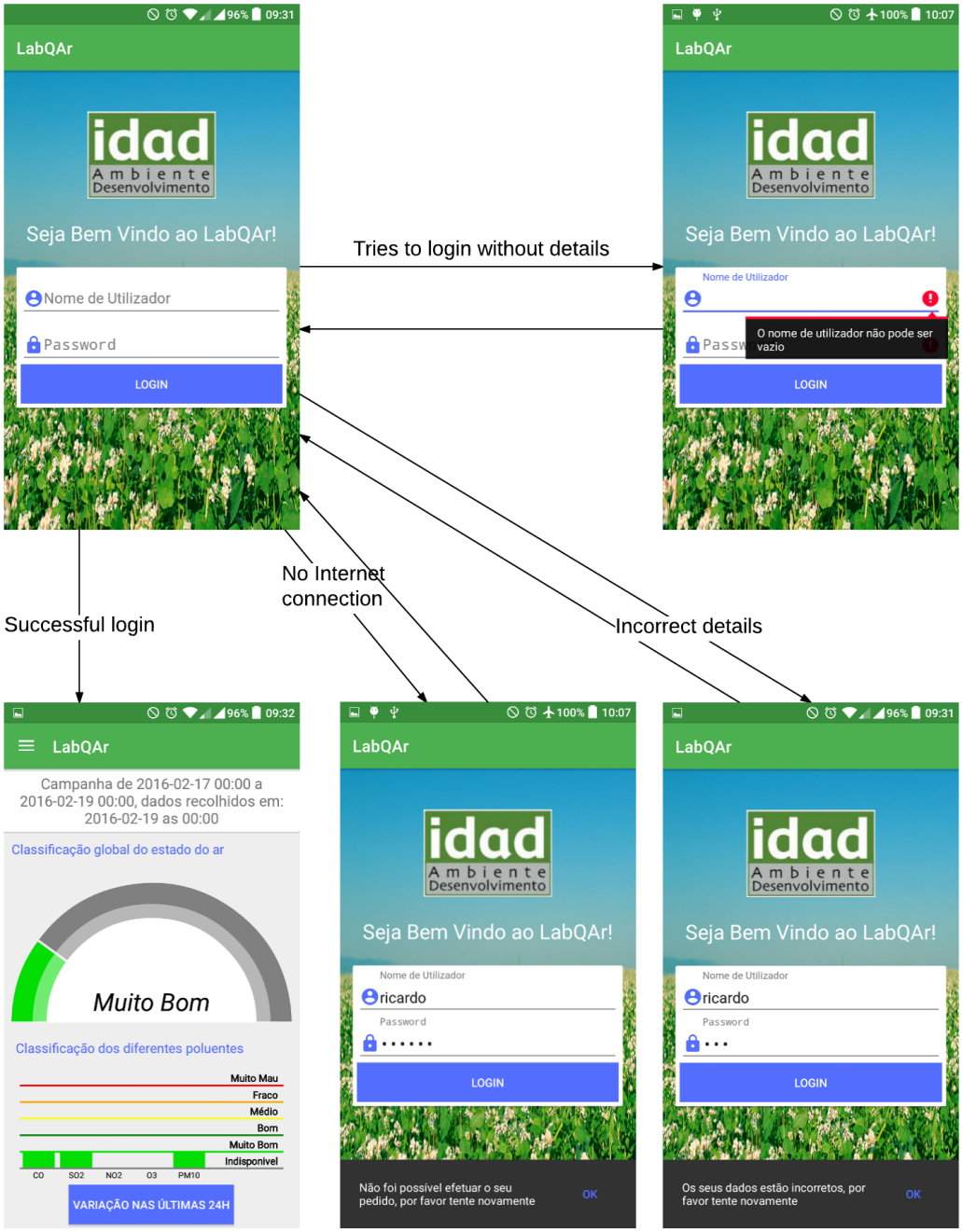


Figure 5.20: Login Screen Storyboard

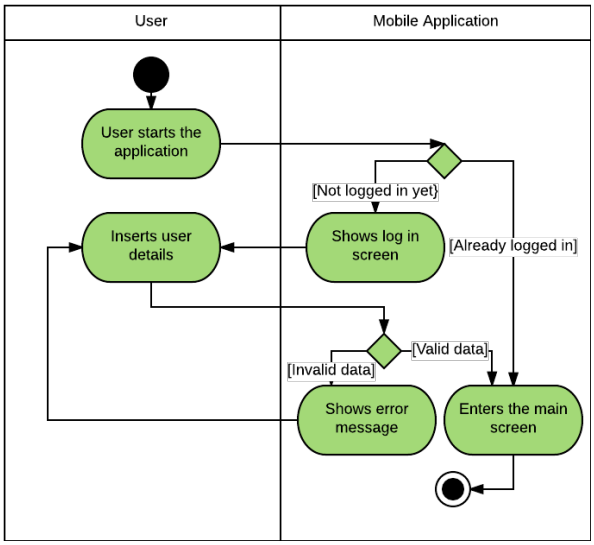


Figure 5.21: Login Screen Activity Diagram

5.4.2.2 Overview Of The Air Quality

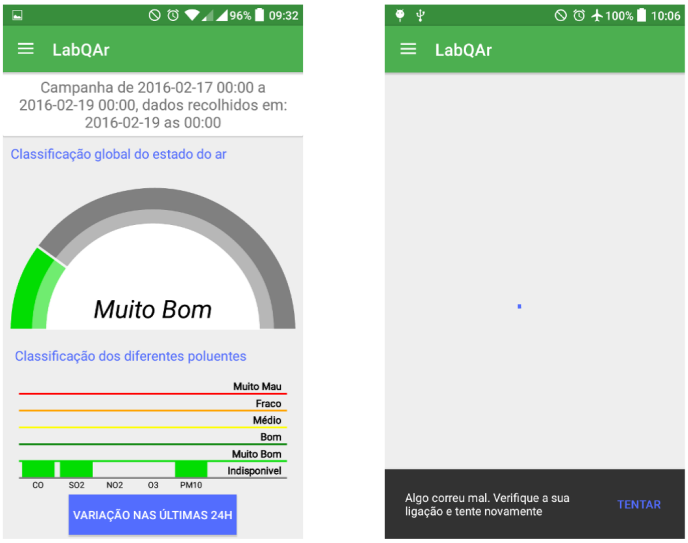


Figure 5.22: Overview Screen Storyboard

In this activity, the application gets the air quality state from the server, through an asynchronous HTTP request with the Retrofit service. If the phone has no Internet connectivity or the request fails to success, it is displayed a Snackbar that informs that the operation was not possible and the user should try again (Figure 5.22)

At the same time the GCM token is sent to the server to be refreshed, if needed.

5.4.2.3 Overview Of The Air Quality in The Last 24H

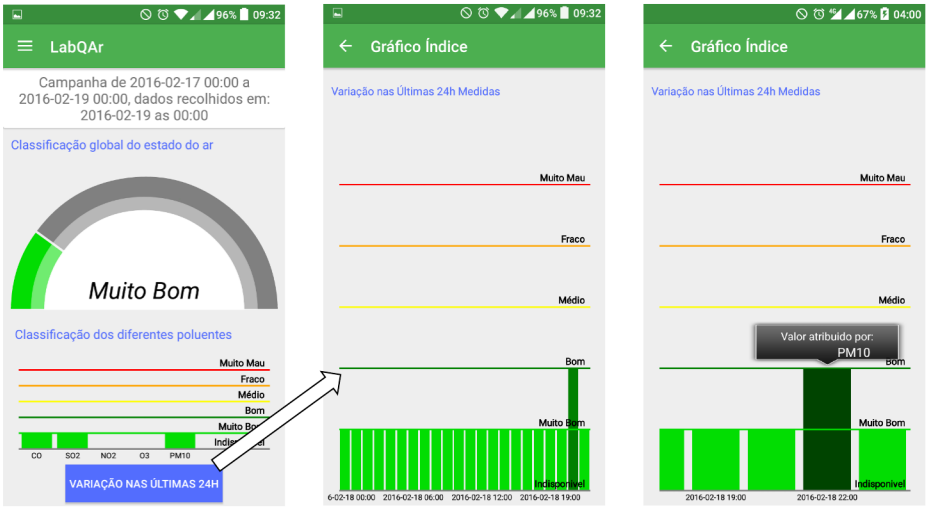


Figure 5.23: Overview 24H Screen Storyboard

To access this feature, the user clicks the button on the homepage. In this activity, the application gets the air quality state for the previous 24 hours from the server, through an asynchronous HTTP request with the retrofit service. It is possible to zoom the chart along both axis and if the user clicks on bar it is presented a custom tooltip indicating the pollutants for the classification verified for that hour. This interaction is represented in the Figure 5.23.

5.4.2.4 Measurements Of Different Parameters

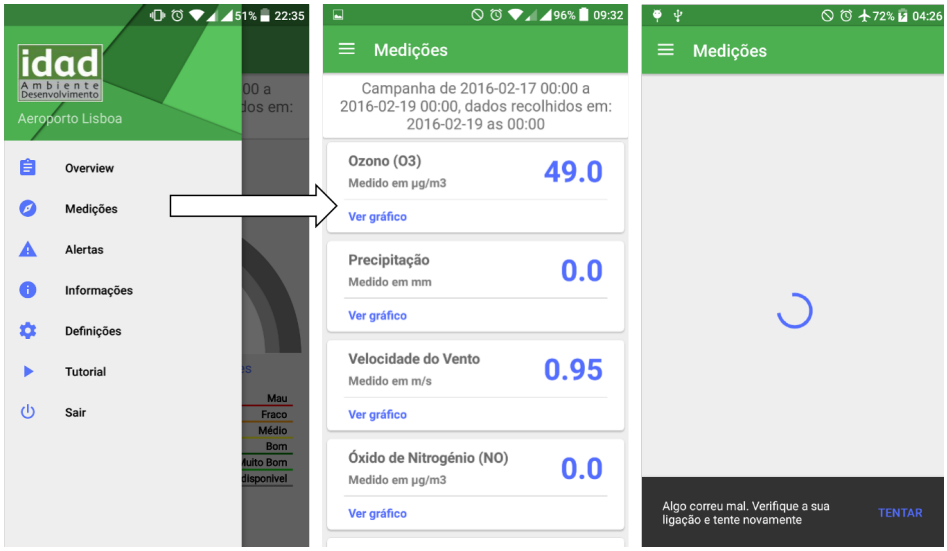


Figure 5.24: Measurements Screen Storyboard

This feature is accessible through the navigation drawer. In this activity, the application gets the value for each parameter from the server, through an asynchronous HTTP request with the retrofit service. The activity uses a RecyclerView to display the data, where each parameter is represented by a CardView. If the phone has no Internet connectivity or the request fails to success, it is displayed a Snackbar that informs that the operation was not possible and the user should try again. This feature is represented by the Figure 5.24.

5.4.2.5 Measurements Of the Selected Parameter In The Last 24H

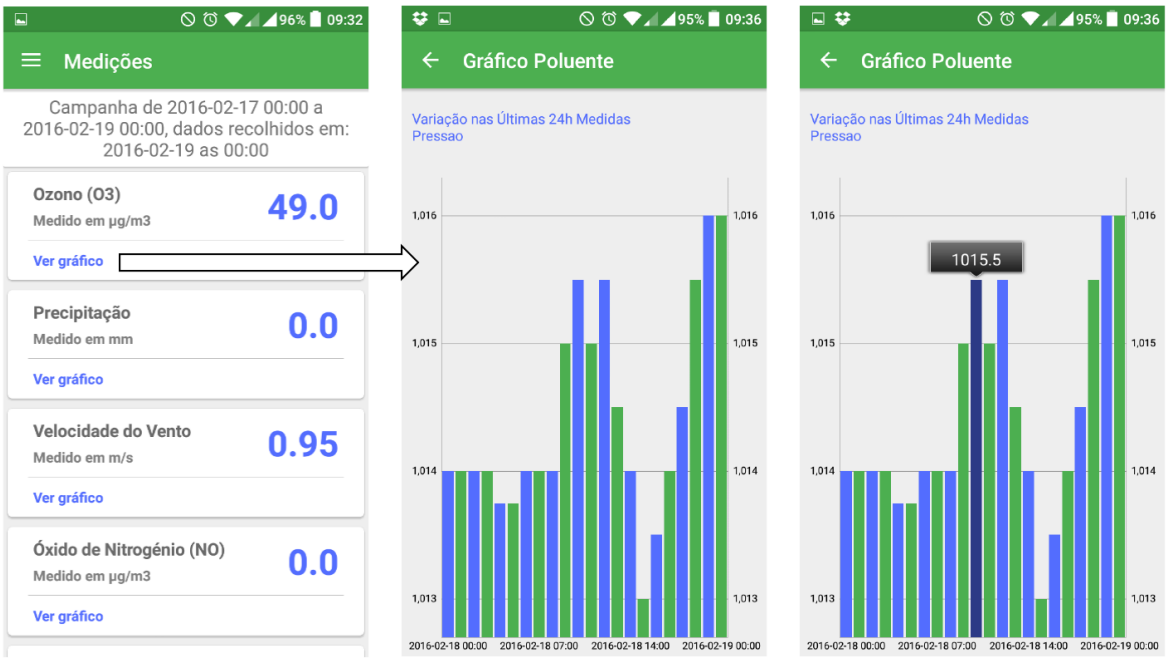


Figure 5.25: Measurments 24H Screen Storyboard

To access this feature, the user clicks the button on the homepage. In this activity, Figure 5.25, the application gets the measurements for the selected parameter during the previous 24 hours from the server, through an asynchronous HTTP request with the retrofit service. In this case it is also possible to zoom the chart along both axis and if the user clicks on bar it is presented a custom tooltip with the precise value measured for that hour.

5.4.2.6 Check Past Notifications

This feature, represented on the Figure 5.26 is also directly available through the navigation drawer. In this activity, the application gets a list of the last set of notifications from the server, through an asynchronous HTTP request with the retrofit service. The activity also uses a RecyclerView to display the data, where each notification is represent by a custom CardView. If the phone has no Internet connectivity or the request fails to success, it is displayed a Snackbar that informs that the operation was not possible and the user should try again.

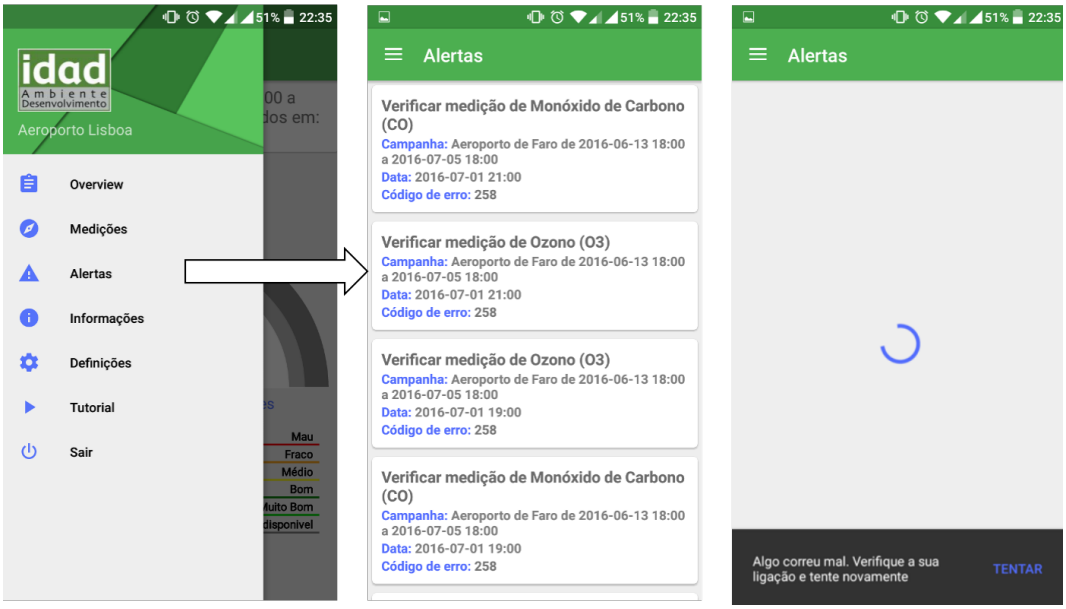


Figure 5.26: Alerts Screen Storyboard

5.4.2.7 Tutorial

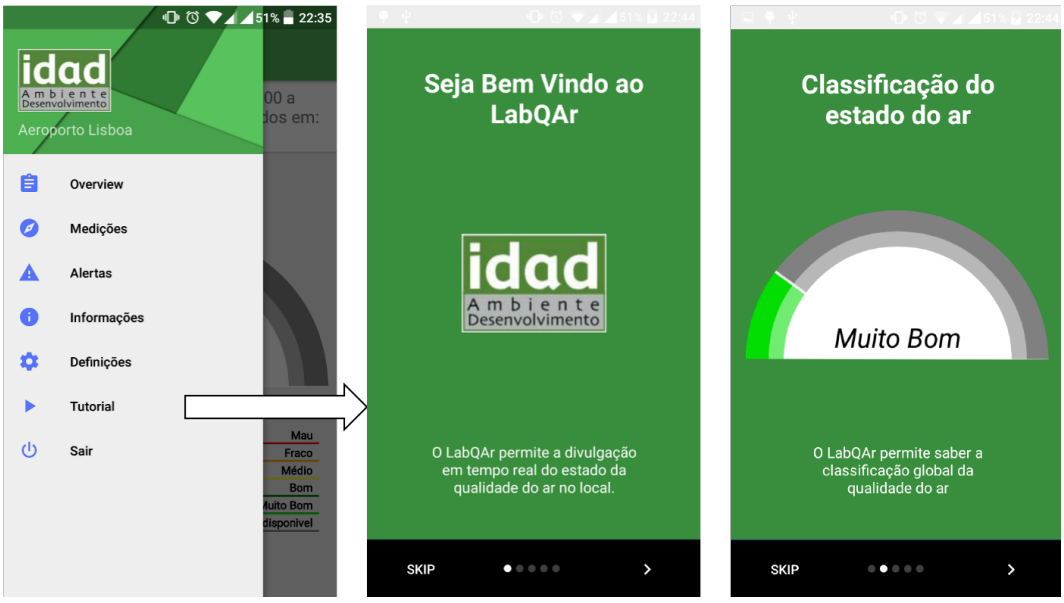


Figure 5.27: Tutorial Screen Storyboard (Not comprehensive)

The tutorial screen is presented to the user automatically the first time the application is executed but it can also be accessed at any time through the navigation drawer. The tutorial is simply a set of information (Figure 5.27) that informs the user about the core features of the application and provides some tips for action that may not be immediately intuitive.

5.4.2.8 Definitions

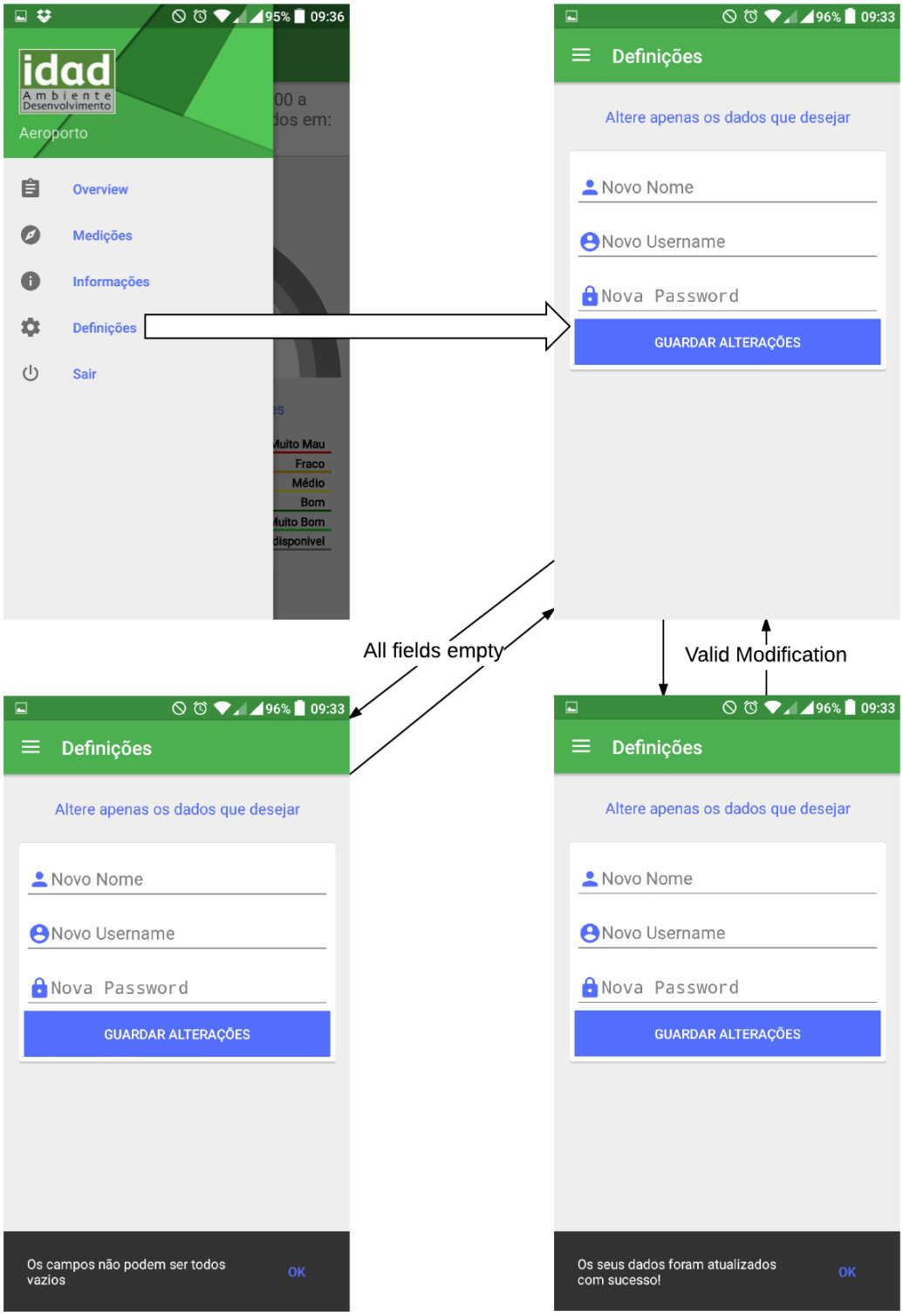


Figure 5.28: Definitions Screen Storyboard

When the user saves the changes, the application will firstly check if all of the details are empty. If so, it shows a snackbar informing the user of that situation. If not, the application will send the request to the server. From there, the application will show a snackbar, either informing the user of the success of the operation or otherwise. If the phone has no Internet connectivity or the request fails to success, it is displayed a Snackbar that informs that the operation was not possible and the user should try again. This logic is represented conceptually in the Figure 5.28.

5.4.2.9 Logout

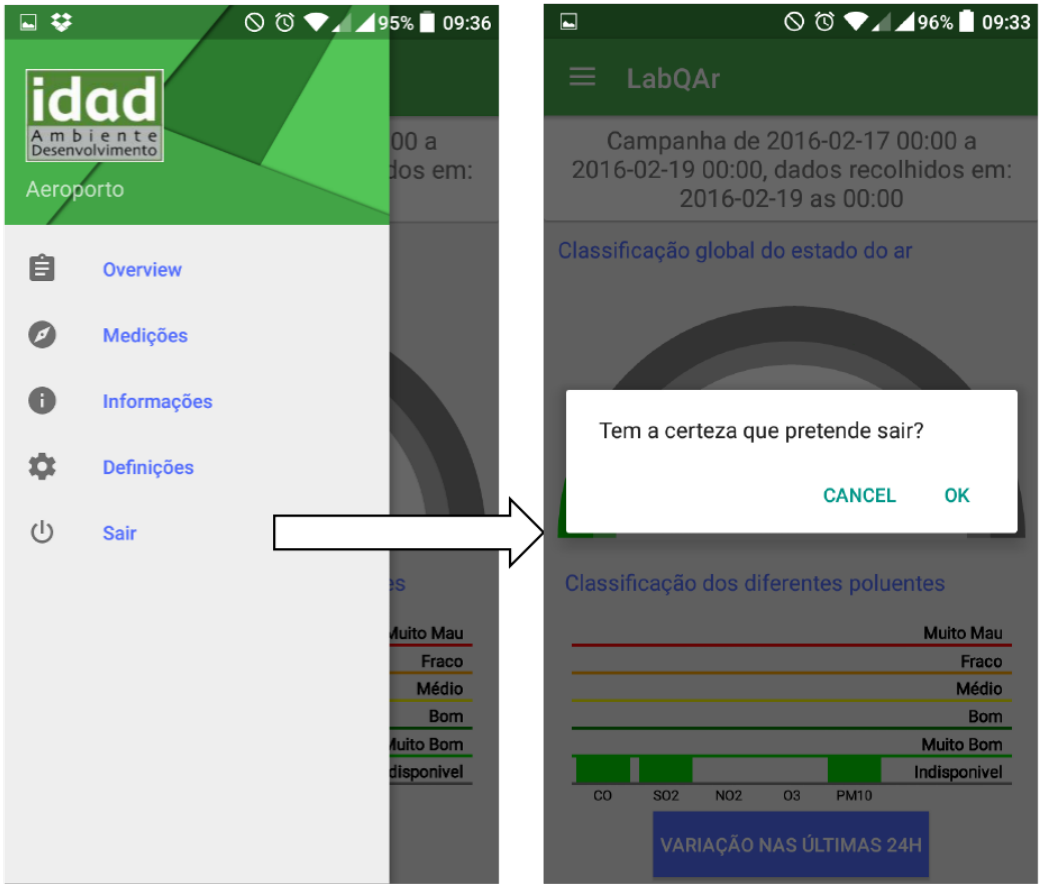


Figure 5.29: Log out Screen Storyboard

When the user selects to log out from the application, he is prompted with a pop up dialog confirming if he really wants to log out, as shown in the Figure 5.29. If he does, the application sends an HTTP request to the server, informing the notification service to stop sending notifications for that device and redirects the user to the login screen, others stays on the same screen.

Chapter 6

System Validation

To validate the system we performed some tests to assess the proper operation of the system, both at a visual and a functional level.

At the functional level, the majority of the validation performed result from errors occurred and fixed during development time and also from the active search of typical errors like, for example, the lack of connectivity in the mobile phone.

The mobile application is also equipped with Crashlytics and Answers kits from the Twitter's Fabric SDK. The first kit is a light weight crash reporting solution which saves information about application crashes and the error occurred as well as some important data about the device where the error occurred. The Answers' kit provides mobile analytics, like the most used features, daily or monthly number or users, etc.

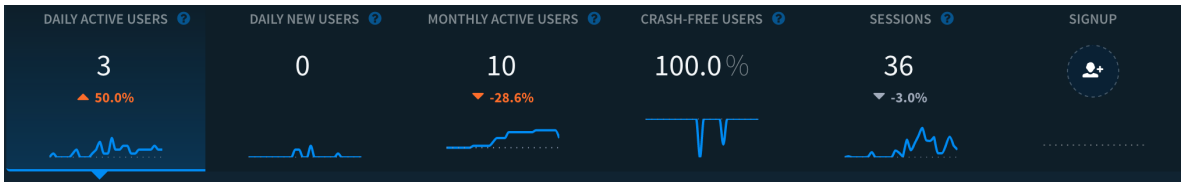


Figure 6.1: Example of Answers' daily analytics

Figure 6.1 shows an example of the analytics' daily digest that can be obtained from the Answers kit, highlighting the number of active users, new users, active users during the last month, number of crash free users and total daily sessions.

6.1 Compatibility Tests

Compatibility Tests are required to ensure that both the web and mobile application work seamlessly across different browsers and mobile devices, respectively.

For the Web Application, the compatibility tests performed passed by using the Website in different browser. The browsers tested where: Google Chrome, Safari, Mozzila Firefox and Vivaldi. In all of them the layouts were correct and the system performed every operation correctly. The only noticeable different was in the animation, which were fluid for every

browser, except for Vivaldi.

Another compatibility concern of the website is that its responsive design allows it to be accessed from a browser in a desktop, a tablet (Figure 6.4) or even smartphones (6.5) while keeping a coherent look. If screen is smaller than a certain resolution, the side bar automatically hides to make room for the other most important components, which is something very familiar in mobile apps.

Figure 6.2: Drawer Hidden.

Figure 6.3: Drawer Visible.

Figure 6.4: Tablet Size Screen

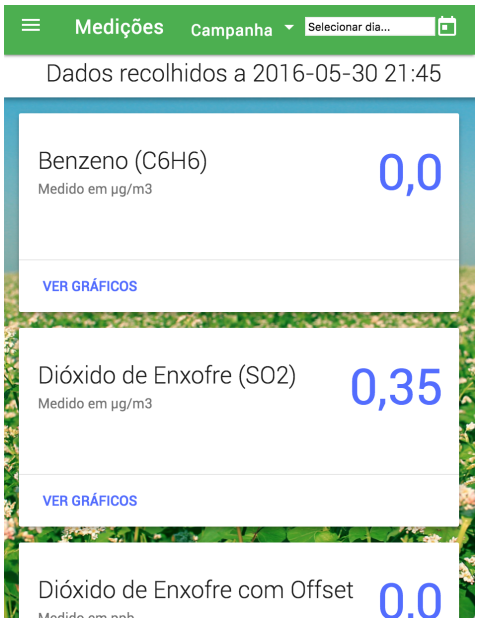


Figure 6.5: Mobile Size Screen

The mobile application was also tested in the different devices, screen sizes and Android versions. In the analysis of the different devices, there are some parameters that should be observed:

- Correctness of the layouts
- Animation support
- Material design compatibility
- Correct functioning of the system

The different screen sizes and Android versions used are listed in the table bellow. The devices chosen covers older devices with older Android versions and smaller screen, recent devices with the latest Android versions and also a Tablet with a bigger resolution.

Device	Android Version	Screen Resolution
Google Nexus S (Smartphone)	4.1.1	480x800
Google Nexus 10 (Tablet)	5.1.0	2560x1600
Samsung Galaxy S6 (Smartphone)	6.0.0	1440x2560

Table 6.1: Devices tested

Figure 6.6: Device 1.

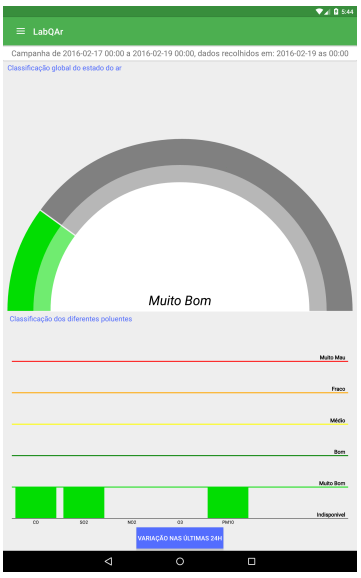


Figure 6.7: Device 2.

Figure 6.8: Device 3.

Figure 6.9: Home screen across the different devices tested.

On the 3 devices tested, the application keeps the desired layout, as it can be seen in the Figure 6.9. Only on the tablet screen with a much bigger screen the components seem a bit small. Also, the page showed in the example above is the one which is more susceptible to

big changes, since the others composed mostly by RecyclerView and full screen charts don't change too much between devices.

All the devices supported the Material Design correctly and all the animations desired. The application also performs the expected features correctly on all of them.

For a better user experience, the layout is also adaptive to the screen orientation allowing, for example, to analyse a chart on landscape (Figure 6.11) mode instead of portrait (Figure 6.10), which typically may be desirable due to the bigger size of the screen.

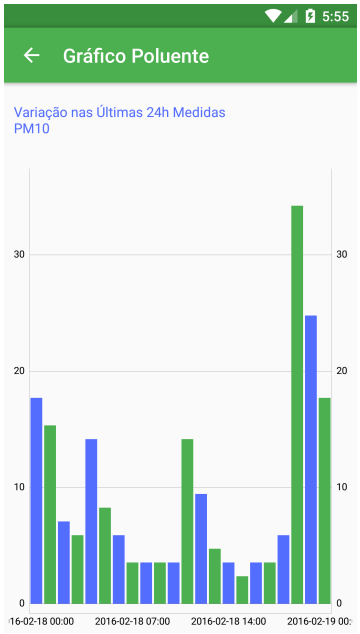


Figure 6.10: Portrait Mode.

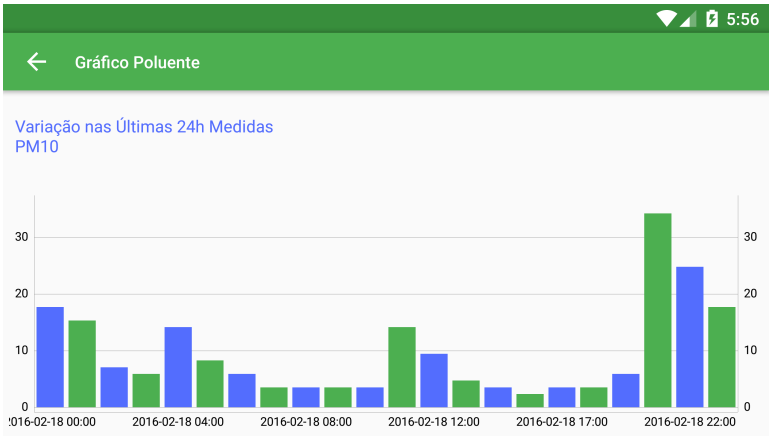


Figure 6.11: Landscape Mode.

Figure 6.12: Chart on different screen orientations.

6.2 Pilot usage

Starting a couple days before and during a campaign started 13th June 2016 until 5th July 2016 the system was subject to a pilot usage to validate the proper operation on utility of the system. In the beginning of the month, the system was tested at first by users from the IDAD who assessed the solution and, shortly after the start of the campaign expanded the testing phase to the interested company. The Figure 6.13, taken from the Answers kit, shows the increase of active users throughout the month of June, both IDAD users and their partners started using the system, respectively.

During this usage, the members from the Institute assessed the accuracy of the data by comparing it with the equivalent data presented by the Atmis software and both the mobile and web solution worked as expected without noticeable flaws.

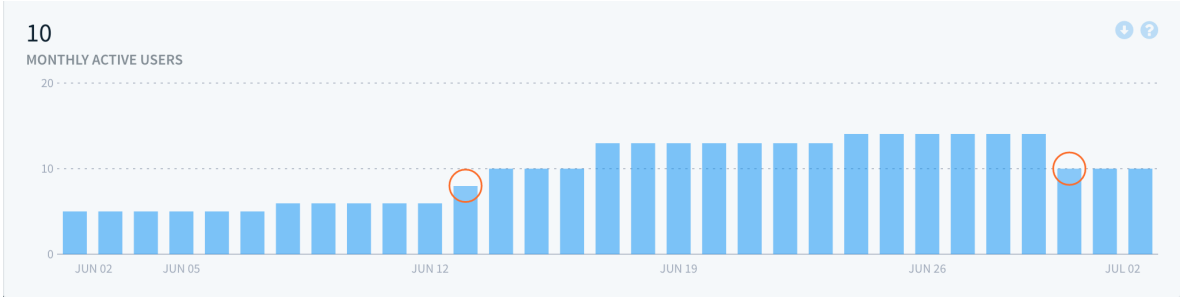


Figure 6.13: Daily Active Users in June

Chapter 7

Conclusion And Future Work

7.1 Conclusion

Before this work, IDAD was not able to provide its partners/customers with live access to the air quality metrics being collected at the mobile lab. The usual procedure was to have the equipment in the van (parked in a remote location) collecting data through specialized sensors and storing them locally.

The practical limitations of knowing if the equipment was operating in normal conditions were mitigated by remote desktop accesses to the computer in the van, but the data was essentially made available to the partners (contracting the air monitoring service) days after being collected.

In this work, we developed an integrated system to enable data upload from the van to a backend server and to present the monitoring data in an updated, structured and friendly way, both through a web site and a mobile application.

The initial target was the development of the mobile application alone, but soon it was clear that a full stack solution would provide a better support and more flexibility. It is possible, for example, to use the web application for the campaigns configuration and customization, extending the concerns beyond the air monitoring aspects.

The full-stack solution required the integration from different technologies, which was an enriching endeavor: a watchdog to extract data from the Microsoft SQL Server at the van; message-based dispatching of the new samples though a cloud Message Broker (RabbitMQ); long term storage in a PostgreSQL database; web site development with the Python-based Django framework; REST API to enable the integration of future and mobile applications; mobile client for Android; deployment of push notifications over Google Cloud Messaging.

With the developed applications and enabling processes, it is now possible to know the state of the air quality in near real time, using the mobile and web applications. It is also possible to create alarms based on data thresholds, and have them trigger notifications to the staff and to the partners' end-users.

IDAD has included the tools developed in this work in their portfolio and is using them in new campaigns, which supports the observation that the proposed goals have been achieved.

7.2 Future Work

As future work, some improvements could be done to the present solution. First, an obvious sequence to the project would be the expansion of the solution for iOS devices. In the current state of the project, an iOS user can still use it through the computer browser or even the mobile browser which, as was shown before, offers an interface quite similar to the mobile application but with disregard for the mobile push notifications. This requires only to build the mobile application since the rest of the system does not require any changes.

Another interesting improvement is the addition of other measuring devices to the system. The work done in this dissertation was done around the data aggregated by the Atmis software, but the LabQAr is equipped with other devices that could possible be integrated in the system.

As a last note, new ways of analysing/visualizing some of the data used in this project could provide new value, both to IDAD and its partners. This would benefit from a joint work from multidisciplinary teams, such as this work.

References

[1] Efeitos genéricos da poluição do ar. Accessed on June 2016. [Online]. Available: http://qualar.apambiente.pt/index.php?page=5&subpage=4

[2] O IDAD. Accessed on June 2016. [Online]. Available: http://www.ua.pt/idad/PageText.aspx?id=9171

[3] Qualidade do ar exterior. Accessed on June 2016. [Online]. Available: http://www.ua.pt/idad/qualidade_do_ar_exterior

[4] Efeitos genricos da poluio do ar. Accessed on June 2016. [Online]. Available: http://qualar.apambiente.pt/index.php?page=5

[5] G. Fekete. Being a full stack developer. Accessed on June 2016. [Online]. Available: https://www.sitepoint.com/full-stack-developer/

[6] M. G. Mendez, *The Missing Link: An Introduction to Web Development and Programming*, ser. First Edition. CreateSpace Independent Publishing Platform, 2014.

[7] M. Hartl, *Ruby on Rails Tutorial: Learn Web Development with Rails*, ser. Third Edition. Addison-Wesley Professional, 2015.

[8] Getting started with rails. Accessed on June 2016. [Online]. Available: http://guides.rubyonrails.org/getting_started.html

[9] L. Teo. Ruby on rails vs php the good, the bad. Accessed on June 2016. [Online]. Available: http://www.leonardteo.com/2012/07/ruby-on-rails-vs-php-the-good-the-bad/

[10] The web framework for perfectionists with deadlines. Accessed on June 2016. [Online]. Available: https://www.djangoproject.com/

[11] Node.js. Accessed on June 2016. [Online]. Available: https://nodejs.org/en/

[12] P. Wayner. 13 fabulous frameworks for node.js. Accessed on June 2016. [Online]. Available: http://www.infoworld.com/article/3064653/application-development/13-fabulous-frameworks-for-nodejs.html#slide3

[13] T. Capan. Why the hell would i use node.js? a case-by-case tutorial. Accessed on June 2016. [Online]. Available: https://www.toptal.com/nodejs/why-the-hell-would-i-use-node-js

[14] R. Meier, *Professional Android 4 Application Development*, ser. Third Edition. Wrox, 2012.

[15] M. Haselmayr. Here's why your business needs its own mobile app. Accessed on June 2016. [Online]. Available: http://www.forbes.com/sites/allbusiness/2014/11/17/heres-why-your-business-needs-its-own-mobile-app/#68e9afe85c76

[16] C. Bonnington. In less than two years, a smartphone could be your only computer. Accessed on June 2016. [Online]. Available: http://www.wired.com/2015/02/smartphone-only-computer/

[17] D. Chaffey. Mobile marketing statistics 2016. Accessed on June 2016. [Online]. Available: http://www.smartinsights.com/mobile-marketing/mobile-marketing-analytics/mobile-marketing-statistics/

[18] Idc: Smartphone os market share 2015, 2014, 2013, and 2012. Accessed on June 2016. [Online]. Available: http://www.idc.com/prodserv/smartphone-os-market-share.jsp

[19] M. Korf and E. Oksman. Native, html5, or hybrid: Understanding your mobile application development options. Accessed on June 2016. [Online]. Available: https://developer.salesforce.com/page/Native,_HTML5,_or_Hybrid:_Understanding_Your_Mobile_Application_Development_Options

[20] M. Fowler, *UML Distilled: A Brief Guide to the Standard Object Modeling Language*, ser. Third Edition. Addison-Wesley Professional, 2013.

[21] R. Carreira, *Atmis 8.0 - Caracteristicas Tecnicas*, ser. First Edition. ISA, 2008.

[22] Celery: Distributed task queue. Accessed on June 2016. [Online]. Available: http://www.celeryproject.org/

[23] Django faq: General. Accessed on June 2016. [Online]. Available: https://docs.djangoproject.com/en/1.9/faq/general/

[24] A. Freeman, *Pro ASP.NET MVC 5*, ser. Fifth Edition. Apress, 2013.

[25] A. Holovaty and J. Kaplan-Moss, *The Definitive Guide to Django: Web Development Done Right*, ser. Second Edition. Apress, 2009.

[26] E. Burris, *Programming in the Large with Design Patterns*, ser. First Edition. Pretty Print Press, 2012.

Appendix

A1: Use cases description

This appendix complements the definition of the use cases specified in the System Requirements section, providing, for each one, the use case description and activity diagram.

- **Check Air Quality Classification**:

Name	Check Air Quality Classification
Actors	Staff Members, Partners
Brief Description	The user checks the quality of the air in the latest data gathered for his campaigns or the last overall data, if the user is a staff member.
Preconditions	The user is logged in the system.
Basic Flow	1. The user accesses the website. 2. The user is presented with the main page, Overview. 3. The website presents a chart with the overall classification and another with the classification of each parameter used.
Alternative Flow	1. The user opens up the mobile application. 2. The user is presented with the main page, Overview. 3. The mobile application presents a chart with the overall classification and another with the classification of each parameter used.

Table A1.1: Check Air Quality Classification use case description

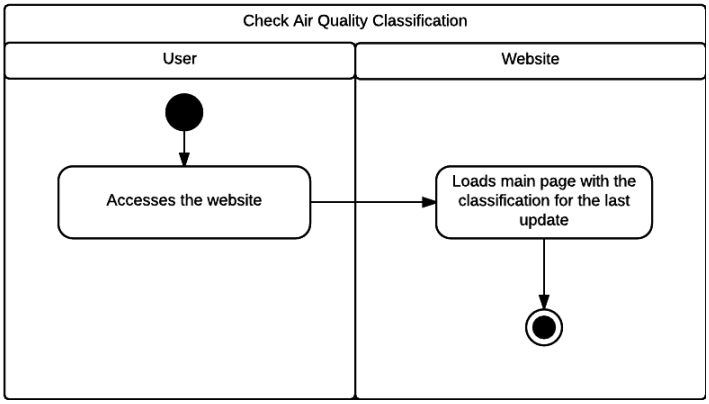


Figure A1.1: Check Air Quality Classification use case activity diagram, web version

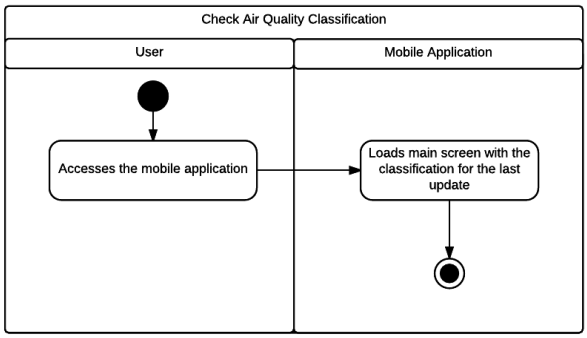


Figure A1.2: Check Air Quality Classification use case activity diagram, mobile version

- **Analyse Air Quality Variation In The Last 24h**:

Name	Analyse Air Quality Variation In The Last 24h
Actors	Staff Members, Partners
Brief Description	The user analyses the variation of the air quality during 24h.
Preconditions	The user is logged in the system.
Basic Flow	1. The user accesses the website. 2. The user is presented with the main page, Overview. 3. The user selects the button "Variação nas ultimas 24H" 4. The website presents a chart with the variation of the air quality during 24h.
Alternative Flow	1. The user opens up the mobile application. 2. The user is presented with the main page, Overview. 3. The user selects the button "Variação nas ultimas 24H" 4. The mobile application presents a chart with the variation of the air quality during 24h.

Table A1.2: Analyse Air Quality Variation In The Last 24h use case description

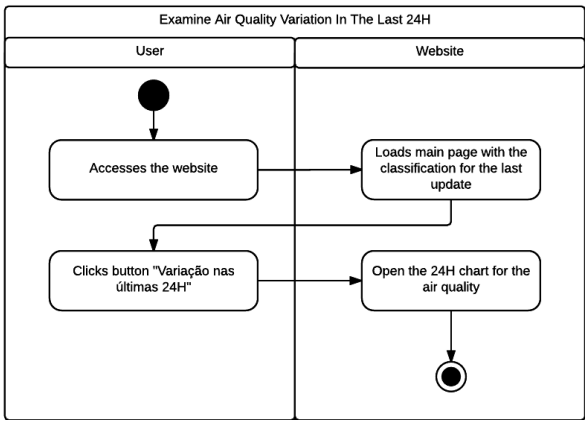


Figure A1.3: Analyse Air Quality Variation In The Last 24h use case activity diagram, web version

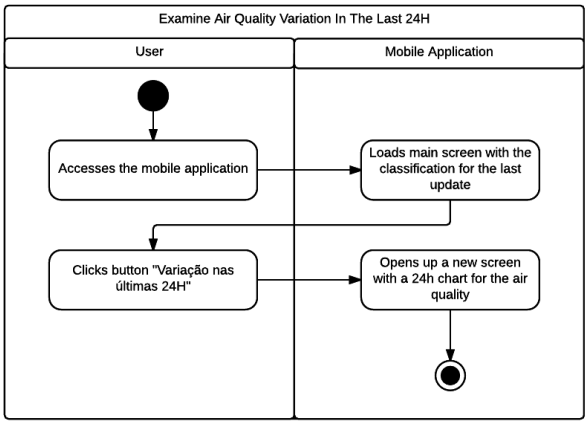


Figure A1.4: Analyse Air Quality Variation In The Last 24h use case activity diagram, mobile version

- **Check Latest Data**:

Name	Check Latest Data
Actors	Staff Members
Brief Description	The user checks the latest data available for him.
Preconditions	The user has a staff account or has at least one campaign associated with him, and is logged in the system.
Basic Flow	1. The user accesses the website. 2. The user is presented with the main page, Overview. 3. The user selects the page Medições. 4. The website presents the data related to the latest data available for the user.
Alternative Flow	1. The user opens up the mobile application. 2. The user is presented with the main screen, Overview. 3. The user selects the button Medições from the main drawer. 4. The mobile application presents the user with the latest data available for the user.

Table A1.3: Check Latest Data

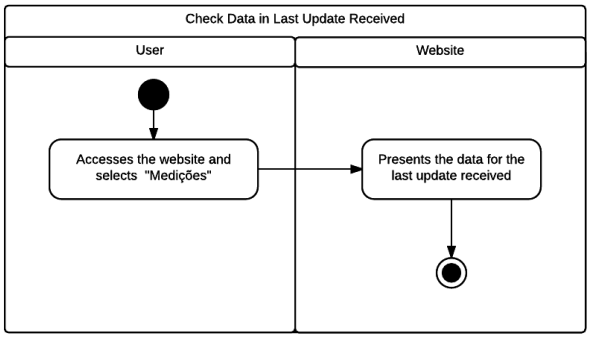


Figure A1.5: Check Latest Data use case activity diagram, web version

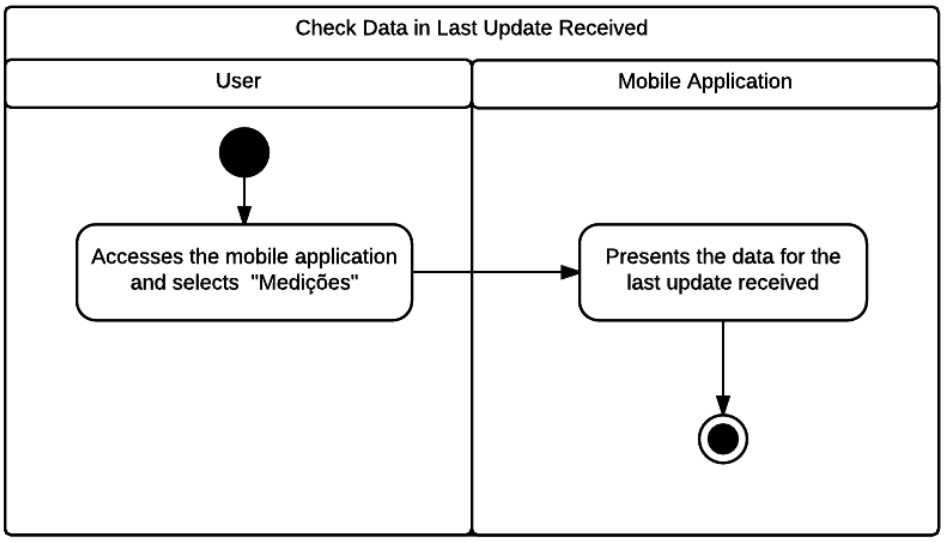


Figure A1.6: Check Latest Data use case activity diagram, mobile version

- **Analyse 24H Variation Of A Parameter**:

Name	Analyse 24H Variation Of A Parameter
Actors	Staff Members, Partners
Brief Description	The user analyses the variation of a given parameter during the previous 24 hours of the latest data
Preconditions	The user is logged in the system.
Basic Flow	1. The user accesses the website. 2. The user is presented with the main page, Overview. 3. The user selects the page Medies. 4. The website presents a list of the latest values for each parameter. 5. The user selects the parameter to examine. 6. The website presents a chart regarding the variation of the selected parameter during the previous 24 hours.
Alternative Flow	1. The user opens up the mobile application. 2. The user is presented with the main screen, Overview. 3. The user selects the button Medies from the main drawer. 4. The mobile application presents a list of the latest values for each parameter. 5. The user selects the parameter to examine. 6. The mobile app presents a new screen with a chart regarding the variation of the selected parameter during the previous 24 hours.

Table A1.4: Analyse 24H Variation Of A Parameter use case description

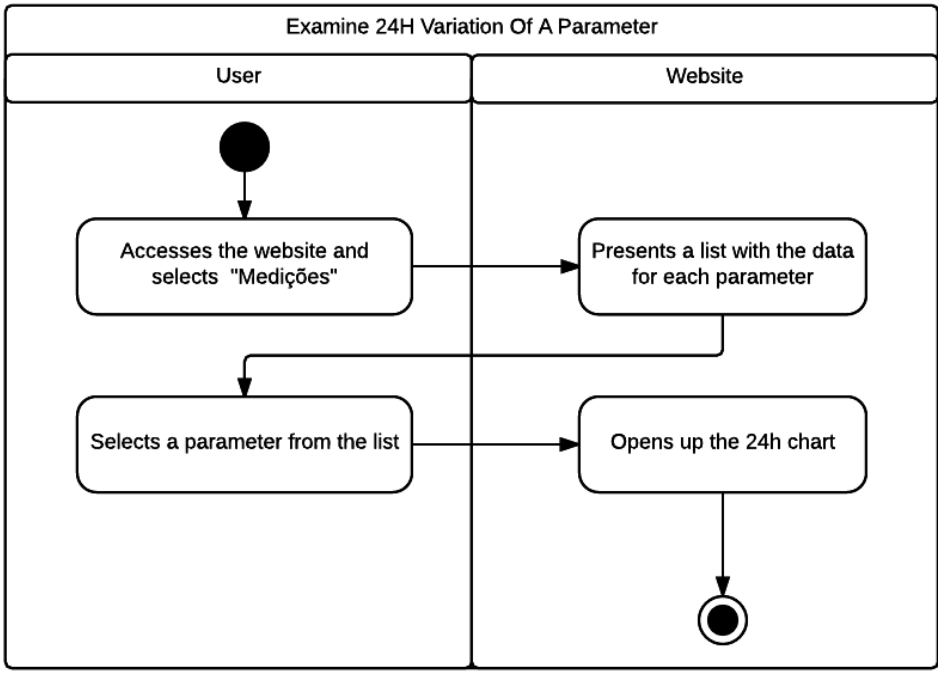


Figure A1.7: Analyse 24H Variation Of A Parameter use case activity diagram, web version

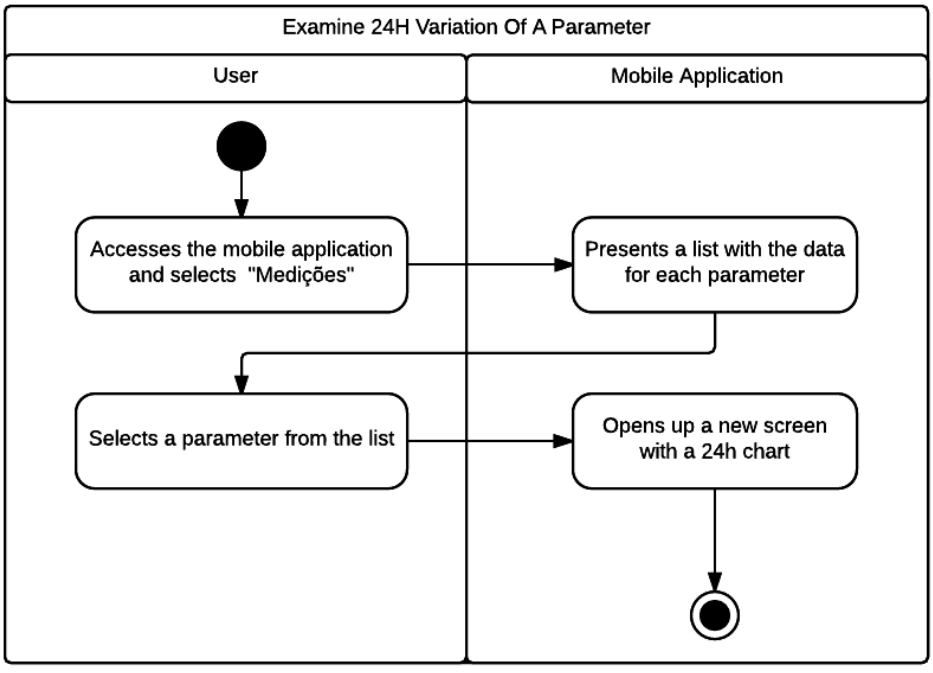


Figure A1.8: Analyse 24H Variation Of A Parameter use case activity diagram, mobile version

- **Analyse Variation Of a Parameter In A Campaign**:

Name	Analyse Variation Of a Parameter In A Campaign
Actors	Staff Members, Partners
Brief Description	The user analyses the variation of a given parameter during a campaign
Preconditions	The user is logged in the system.
Basic Flow	1. The user accesses the website. 2. The user is presented with the main page, Overview. 3. The user selects the page Medies. 4. The website presents a list of the latest values for each parameter. 5. The user selects the parameter to examine. 6. The website presents a chart regarding the variation of the selected parameter during the previous 24 hours. 7. The user presses the button Campanha and its presented with the variation of the parameter during the campaign.

Table A1.5: Analyse Variation Of a Parameter In A Campaign use case description

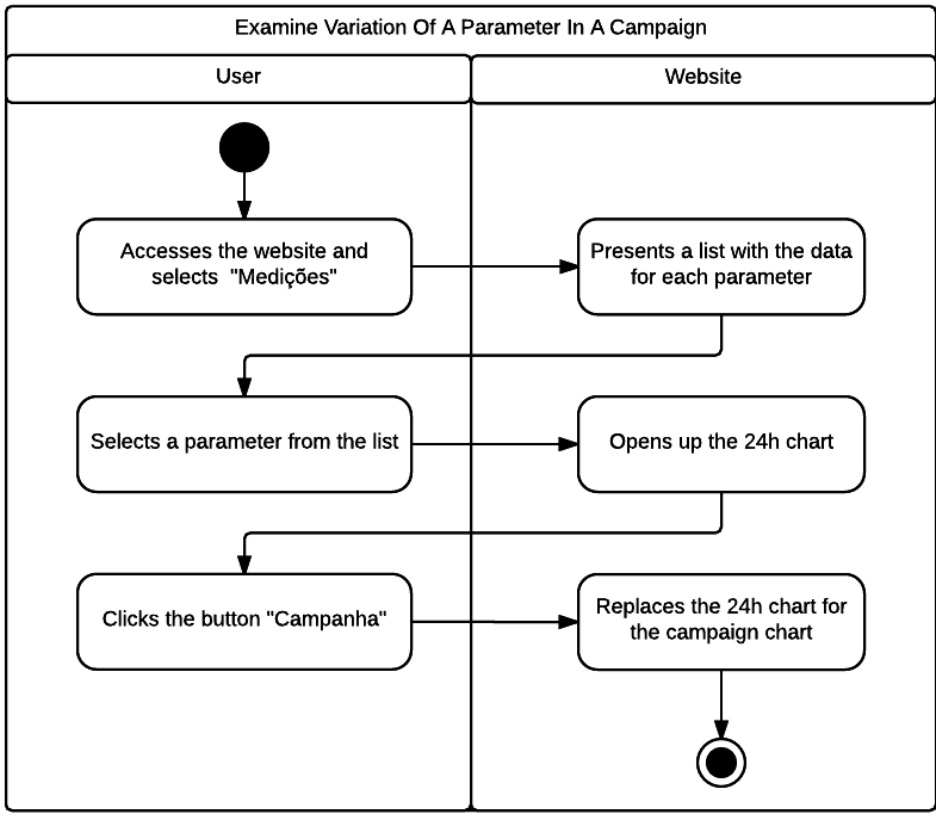


Figure A1.9: Analyse Variation Of a Parameter In A Campaign use case activity diagram

- **Select Campaign**:

Name	Select Campaign
Actors	Staff Member, Partners
Brief Description	The user selects the campaign he wants to check.
Preconditions	The user is logged in the system and, the user is a staff member or has at least one campaign associated with his account.
Basic Flow	1. The user accesses the website. 2. The user clicks the dropdown button named Campanha and choses the campaign to check. 3. The website updates the page with the data from the chosen campaign.

Table A1.6: Select Campaign use case description

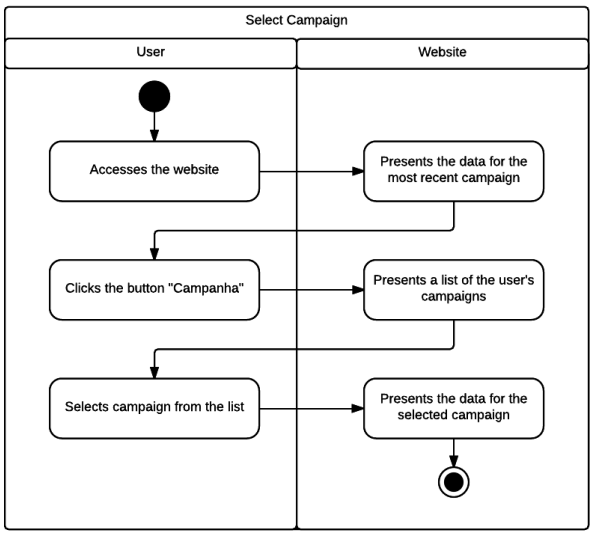


Figure A1.10: Select Campaign use case activity diagram

- **Browse a Specific Date**:

Name	Browse a Specific Date
Actors	Staff Member, Partners
Brief Description	The user selects a date from a calendar.
Preconditions	The user is logged in the system and, the user is a staff member or has at least one campaign associated with his account.
Basic Flow	1. The user accesses the website. 2. The user clicks the calendar button next to Selecionar dia.. and chooses the day to check. 3. The website updates the page with the data from the chosen day.

Table A1.7: Browse a Specific date use case description

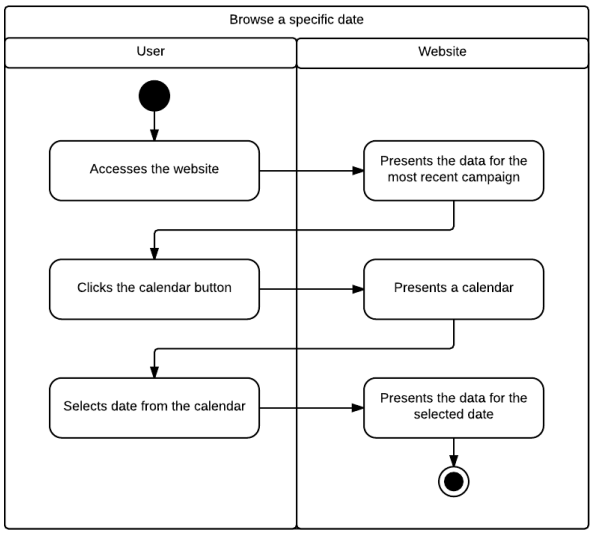


Figure A1.11: Browse a specific date use case activity diagram

- **Check Detected Errors and Alarms/Received Notifications**:

Name	Check Detected Errors and Alarms/Received Notifications
Actors	Staff Member, Partners
Brief Description	The user checks previous received alarms and errors or received notifications.
Preconditions	The user is logged in the system and has at least one error or alarm associated with his account.
Basic Flow	1. The user accesses the website. 2. The user selects the page Alertas. 3. The website presents the data related to the past alarms received by the user.
Alternative Flow	1. The user starts the application. 2. The user opens the main drawer and selects the button Alertas. 3. The application shows a screen with the past notifications received by the user.

Table A1.8: Check Detected Errors and Alarms/Received Notifications

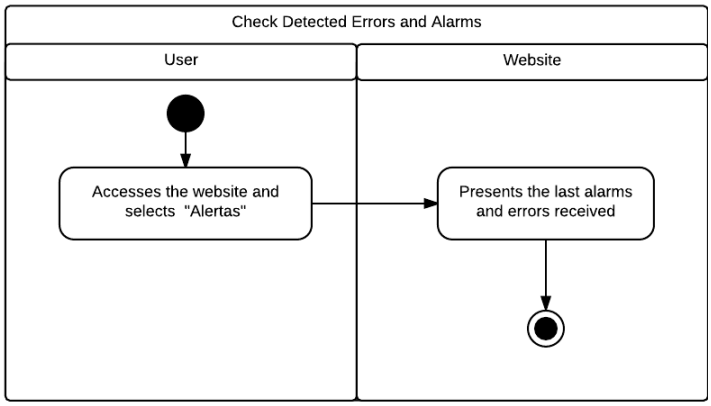


Figure A1.12: Check Detected Errors and Alarms/Received Notifications, web version

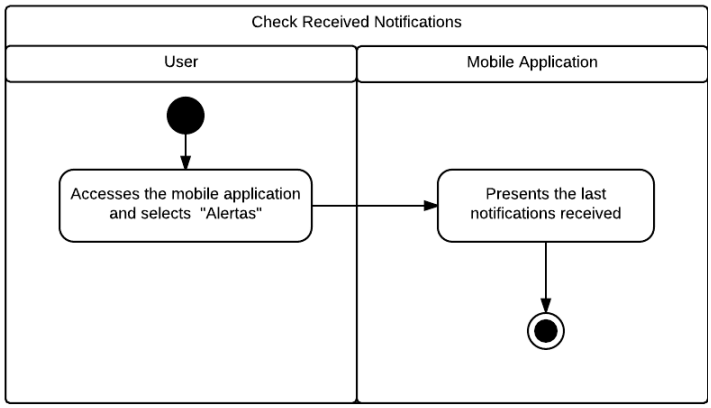


Figure A1.13: Check Received Notifications, mobile version

- **Authenticate**:

Name	Authenticate
Actors	Staff Members, Partners
Brief Description	The user authenticates to the system.
Preconditions	The user has a valid account and is not logged in the system yet.
Basic Flow	1. The user accesses the website. 2. The website redirects the user to the login page 3. The user enters his account details and submits them. 4. If the details are correct, the user is logged into the website.
Alternative Flow	1. The user opens up the mobile application. 2. The application prompts the user with the login page. 3. The user enters his account details and submits them. 4. If the details are correct, the user is logged into the mobile application.

Table A1.9: Authenticate use case description

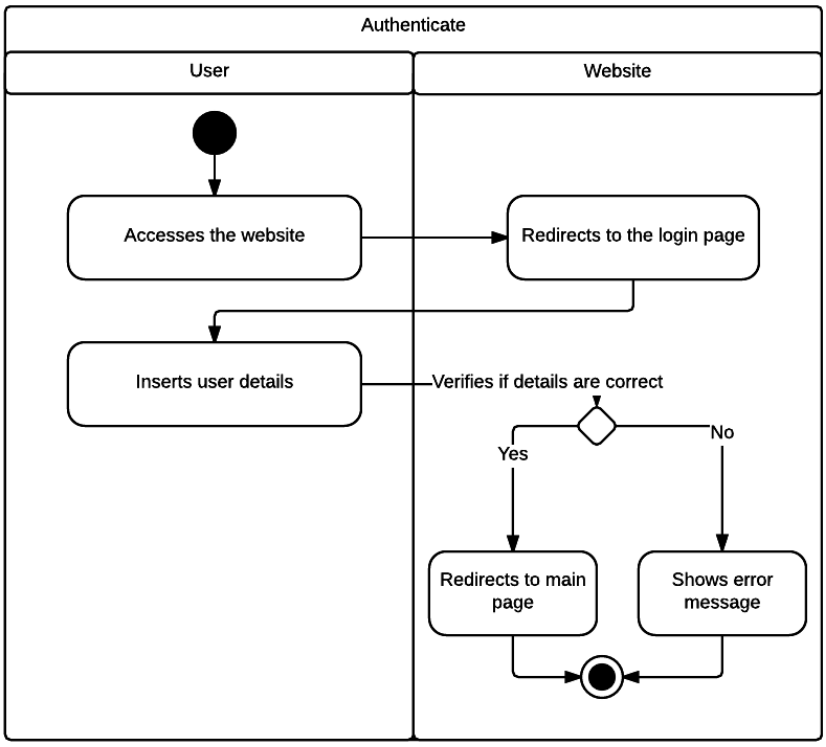


Figure A1.14: Authenticate use case activity diagram, web version

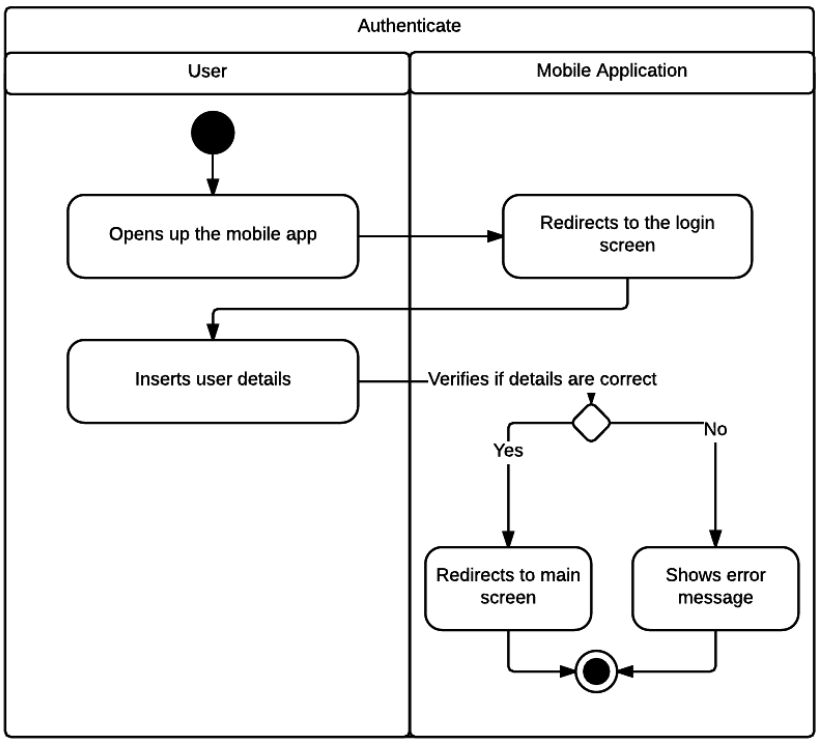


Figure A1.15: Authenticate use case activity diagram, mobile version

- **Manage Account Data**:

Name	Manage Account Data
Actors	Staff Members, Partners
Brief Description	The user edits his account related details.
Preconditions	The user is logged in the system.
Basic Flow	1. The user accesses the website. 2. The user selects the page Definies de Conta. 3. The user enters his updated account details and submits them. 4. If the updated details are valid, the changes are saved to the system.
Alternative Flow	1. The user opens up the mobile application. 2. The user opens the main drawer and selects the button Definies. 3. The user enters his updated account details and submits them. 4. If the updated details are valid, the changes are saved to the system.

Table A1.10: Manage Account Data use case description

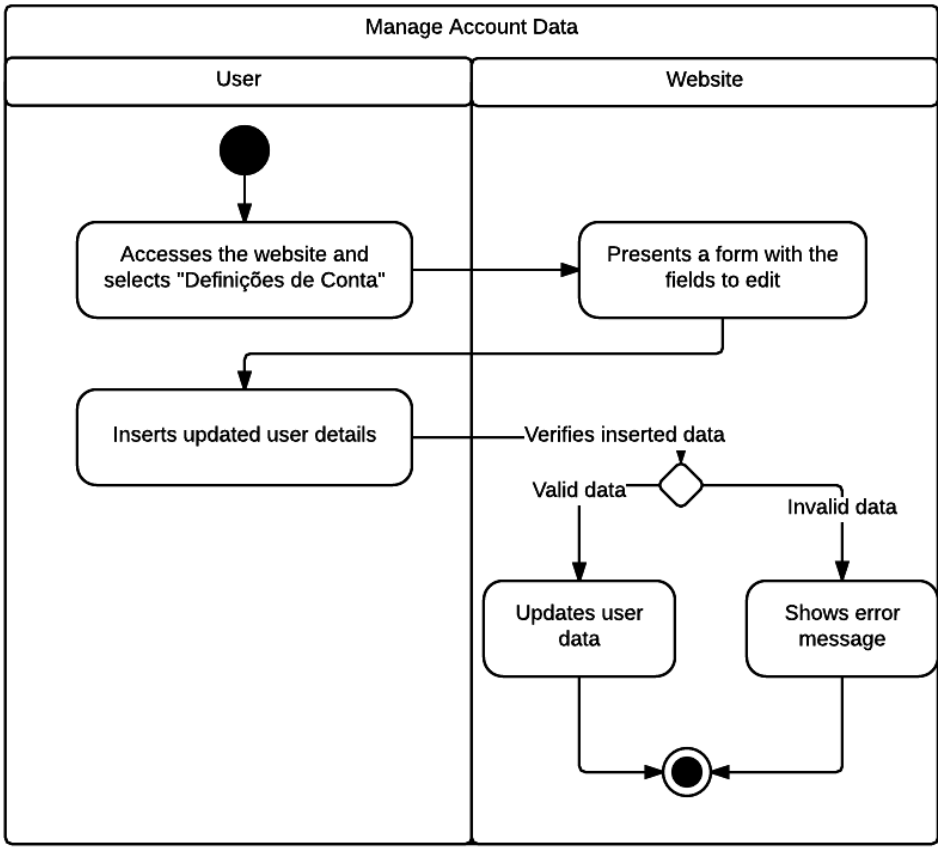


Figure A1.16: Manage Account Data use case activity diagram, web version

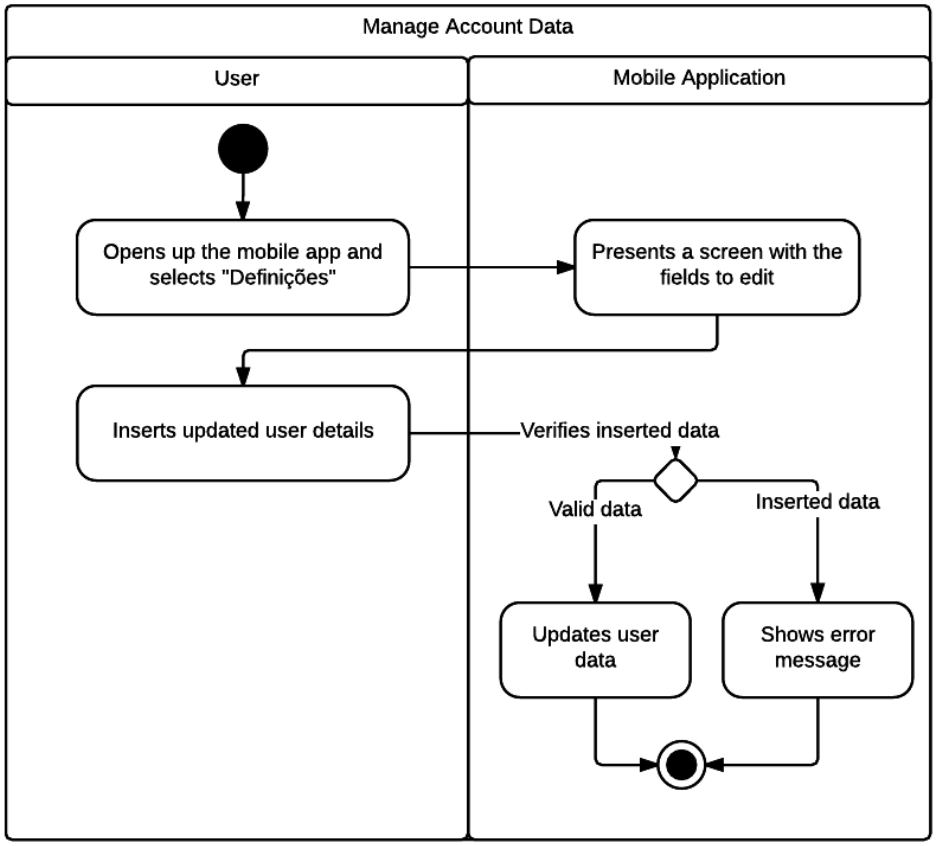


Figure A1.17: Manage Account Data use case activity diagram, mobile version