Master in Informatics Engineering
Internship
Final report

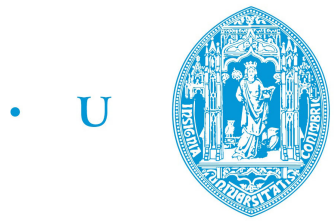
Desenvolvimento de módulos para plataforma Intranet especializada em contexto universitário

Diogo Gonçalves Costa
diogogc@student.dei.uc.pt

DEI Supervisor:
Prof. Dr. Filipe Araújo

Streamline Supervisor:
Eng. Francisco Maia

Date: 03 September 2017



FCTUC FACULDADE DE CIÊNCIAS
E TECNOLOGIA
UNIVERSIDADE DE COIMBRA

Abstract

The current web platform Backoffice from the Department of Electrical and Computer Engineering (DEEC) of University of Coimbra is used by all the people in the department, mostly for academic purposes. However, it has been noted that it has some flaws, mostly in terms of design and code structuring. It also does not meet the current necessities of the department.

Therefore, in this document we present the planning of a new backoffice that will replace the current one. To do that, we have defined new requirements and a new architecture, extending upon the old database for data consistency and backwards compatibility.

We will show how we make use of recent technologies such as Node.js, the LoopBack framework and AngularJS to provide a rich and backwards-compatible experience to all of the students and professors of DEEC. We also make use of other recent web technologies.

Keywords: Web application; Backoffice; LoopBack framework; AngularJS; Node.js

Acknowledgments

This was a long year of constant work that would not be possible without the help and support of some people.

First, I would like to thank my supervisors Prof. Dr. Filipe Araújo for his precious advice and Eng. Francisco Maia for the opportunity of developing a platform to be used by an academic community.

Second, I would like to thank all my friends that really supported me through this process, in special a big thanks to João Ricardo Lourenço for revising this document.

Third, to my family, I would like to thank them for giving me the opportunities in life and for always being there when I need them.

Last, but not least, to my girlfriend, that despite the long distance, can always bring a big smile to my face and motivation to continue. Thank you!

Contents

Acronyms

API - Application programming interface

CSS - Cascading Style Sheets

DEEC - *Department of Electrical and Computer Engineering*

DPI - *Dots per inch*

HTTP - Hypertext Transfer Protocol

HTTPS - Hyper Text Transfer Protocol over SSL

JSON - JavaScript Object Notation

LDAP - Lightweight Directory Access Protocol

MVC - Model–view–controller

ORM - Object Relational Mapping

REST - Representational state transfer

RFID - Radio-Frequency IDentification

SPA - Single Page Application

TCP - Transmission Control Protocol

URI - Uniform Resource Identifier

URL - Uniform Resource Locator

UUID - Universally Unique IDentifier

XHR - XML HTTP Request

XLSX - Microsoft Excel Open XML Spreadsheet

WWW - World Wide Web

List of Figures

List of Tables

1 Introduction

This report represents the work done along this year in the the context of an internship at Streamline [4]. In this chapter we will begin by giving the context of the project, explaining the motivation and defining the goals we want to achieve. Finally we outline the structure of this document.

1.1 Context

Streamline currently provides a set of system information services to the Department of Electrical and Computer Engineering (DEEC) of University of Coimbra. Included in those services, is the DEEC Backoffice platform [5] (we will refer to it as the **old backoffice / old platform / old system**). This platform was developed in its major part by another company that then transferred its rights to Streamline. The web platform is used by every person in the department. It is mainly used for academic purposes (dissertation management, view information) and is also used by the secretary and system administration to perform administration and some maintenance tasks.

This internship's main goal is to develop a new backoffice from scratch to replace the current one. This was a decision made by Streamline and is the reason this project exists. We present in this report the steps taken to come up with new requirements that reflect the necessities of the department and the architecture that will power its functionality. All of this by taking advantage of new technologies and good practices. This includes the analysis of the old platform to identify its flaws and necessities. A main concern of the project is backwards-compatibility, thus leading us to find ways to keep the new system side-by-side with the old data.

The new bacoffice will be called MyDEEC. This was decided in conformity by Streamline and DEEC directors.

1.2 Motivation

When an old software needs new features to be implemented, it can become an extremely hard task, and sometimes even harder than developing a new software. Reasons for that are the lack of code knowledge (lack of documentation, outdated documentation), code not well structured (no separation between business and view logic for example). Outdated and deprecated technologies that compromise the normal system operation and security are also an obstacle to new development.

The old backoffice is included in that group of unmaintainable software. It was its many flaws (both functional and non functional) that led to the Streamline's decision of developing a new backoffice.

One of the motivations for developing a new backoffice from scratch was simple: using well known frameworks to develop the software will help future developers maintain the code by following the rules and patterns defined by those frameworks. One of the major problems when the old backoffice was developed was the lack of a framework. For a project of this size that is always in constant change, it makes sense to make use of existing (and

stable) frameworks, this will ensure that features can be implemented faster and are more maintainable.

There are many features not used in the old backoffice. But most important are the features that are not included but are in demand. Such features include a card access management system, key request system, a new authorization mechanism, new dissertations management system, and many other small features as we will see. Almost all the new features to be implemented are new or are a redefinition of an old feature.

Other problems in the old platform include the fact that the interface of the current platform is not modern and not responsive [6] (adjusts to various screen sizes). Various security issues where also detected in the old platform and must be adressed in the new backoffice.

To end our motivation, we also must state the fact that the old backoffice does not provide an Application Programming Interface (API). For example right now, if Streamline decided to develop a mobile application, the server code would not be prepared to communicate with the mobile application. This is because the server logic directly renders the views and there is no abstraction of server logic. This is another point that motivates the need of developing a new platform.

1.3 Goals

As said before, the main goal of this project is to develop a new backoffice that will replace the old one. We will make extensive use of modern technologies but have to make sure we use them wisely to produce one software that is organized and maintainable. Those technologies alone do not solve our problem. They will sure make development much easier but problems will arise and we have to make sure we implement all the requirements.

Furthermore, we want to maintain compatibility with the old backoffice because both platforms will be running at the same time (while needed features from the old backoffice are not implemented in the new one). We have to solve the problem of data consistency between both platforms and make sure that we can use the data from the old backoffice in the new one (user accounts and information, department information, already existing dissertations). Information cannot be lost nor modified in the migration process.

In terms of performance, we want to make a platform that can support a large amount of concurrent users and has a delivery of content with minimum latency and maximum availability. Although the department does not have a large amount of users, it is good practice to plan the software to scale if needed. The software must also be modular by allowing new developers to easily create new modules.

To achieve all these goals, we first have to analyze the current platform. This includes its available features, architecture and its flaws.

Having gathered information about the system, and considering that the platform is almost 10 years old, we will have to analyze the necessities that are not currently fulfilled and specify the new requirements. With new requirements, we must design an architecture for the platform and implement and test it.

1.4 Document organization

This document is organized in the following chapters:

- **Chapter 2: State of the Art** - In this chapter we present information related to web applications and technologies that we must use. We also compare and select other technologies.
- **Chapter 3: Planning** - In this chapter we will present the work done on the first half of the project and the planning for the second half. This includes estimates and risk management.
- **Chapter 4: Study of the current platform** - We present the features and architecture of the current DEEC Backoffice.
- **Chapter 5: Requirement analysis** - In this chapter we present use cases, their requirements and a an overview of the overall aspect of the application.
- **Chapter 6: Architecture** - In this chapter we will explain in detail how the platform will be structured and how components will be integrated.
- **Chapter 7: Implementation** - It is the chapter were we explain how we implemented the architecture and the problems we have faced while trying to achieve it.
- **Chapter 8: Testing and results** - In this chapter we present how we tested and validated or software. This includes functional, performance and security testing. To end this chapter we also show some usage statistics of the new platform.
- **Chapter 9: Conclusions** - In this brief chapter, we give our final thoughts on the work done.

2 State of the art

Before diving into the planing of the new system, it is necessary to know the technologies that will be used. And then combine it with the knowledge that will be obtained analysing the current deployed system and come with a new solution.

We will first present the background in web applications, then summarise the technologies that must be used in this work, and last, present and select some technologies that we will be using to develop and deploy the new system.

2.1 Web applications background

The Beginning

When in 1991, the World Wide Web [7] opened to the public, several companies started to launch their own websites. At this time, only Hypertext Markup Language (HTML) [8] was used. It is still used today as the base for every website. HTML is a markup language that is utilized to structure the contents of a web page. Text, images, sound and other types of media can be loaded to a page using HTML tags [9].

Technologies

At this time, serving dynamic web pages was hard to achieve because there were no standards or suitable languages to handle server logic. That was until 1995, when languages such as Java [10] and PHP [11] appeared. This was just the beginning of backend development. Since then a vast amount of backend languages has been released with more and more features.

When it comes to the frontend, two of the largest technologies that are broadly used today were also introduced in the 90's decade. They are the Cascading Style Sheets (CSS) [12] and JavaScript [13].

- **CSS** - A language that is used to change the appearance of a web page. That change is achieved by assigning an ID and/or CLASS to an HTML tag. Then, using the CSS language, we can define the style for each ID and CLASS (styles can be also defined to tags, this way all tags will have that style). It is important to note that an ID is supposed to identify only one element on the page in contrast to a CLASS that can identify various elements. A simple example of the CSS syntax is available bellow.

```
/*style for elements assigned with id example-id-1*/
#example-id-1{
    background-color: red;
    color: blue;
    width: 150px;
}

/*style for elements assigned with class example-class-1*/
.example-class-1{
    font-family: Verdana;
```

```
}

/*this is the style for an HTML tag, in this case
the body of the document*/
body{
    background-color: white;
}
```

- **JavaScript** - A scripting language that is generally used on the client side to enable dynamic and interactive web pages. It can also be run in server-side through a JavaScript engine (see section 2 of this chapter). JavaScript can modify the elements and styles of the page. Styles are changed by accessing the CSS attributes of the elements. Here is an example JavaScript code:

```
document.body.style.backgroundColor = 'red';

alert('This is an important message!');

for(var i = 1; i < 10; i++){
document.getElementById('example-id-1').innerHTML = 'The
    ↪ text of this element has changed ' + i + ' times.';
}
```

Web browsers and security

Various web browsers were released during the decade of 90's, being the first developed by Tim Berners-Lee and named WorldWideWeb. Other web browsers like Internet Explorer and Netscape were also released later on this period. The spreading of the WWW enabled companies to expand their business to the Internet, thus the number of e-commerce platforms have greatly increased.

In 2000, encryption was added to the HTTP protocol, in the form of HTTPS (HTTP over TLS) [14]. This empowered companies with a much more secure communication with their customers. With HTTPS, clients can verify the authenticity of the website though certificates. By encrypting all the communications, man-in-the middle attacks are nearly impossible.

Devices and the modern web

Given the fact that the devices used to access web sites have been having their processing power increased, most of the presentation logic has been transferred from the servers to the clients. That means that the servers only handle the business logic and generally expose the functions through an API. Passing the presentation tasks to the client has no arm to the business logic and does not compromise the data if correctly done.

The introduction of HTML5 [15] was a big step. It enabled web pages to use audio, video and graphics drawing without the use of external plug-ins. It also added new tags that enabled the development of more semantic pages (this is important for search engines).

2.2 Summary of the technologies that must be used

Here we are going to present the technologies that we must use. These technologies are being strongly utilized by Streamline and the system to be developed must use them in order to provide a coherent and homogeneous set of services.

Currently, they use technologies such as **Node.js** [16], **LoopBack** [1] and **AngularJS** [17]. Next, we will summarize those technologies. They are the base for what we are going to build so we must be familiar with all the features they can offer and what obstacles we can face.

2.2.1 Node.js

Node.js [16] is an open-source runtime environment that interprets JavaScript code. It is based on Chrome V8 engine [18] by Google. Node.js is cross-platform and can either run server or client applications. Its main advantage is that it has asynchronous input and output and is event driven. That means that the development of real-time scalable applications is much easier. There is no need to maintain an asynchronous workflow with threads. Node.js will handle that for us and every time there is some sort of input, it will throw an event that can be handled through a callback function in the main code.

Node.js was first introduced in 2009, and since then, it has been updated several times, being now in version 7. The use of this technology has been increasing every year [19] by developers and is being adopted by several companies such as eBay, PayPal [20] or LinkedIn [21].

Node.js functionality can be extended by installing new modules. Thousands of them exist for Node.js. Even whole application frameworks are available. Such is the case of LoopBack that we will present in the next section.

The package manager used for installing new modules is npm [22]. In Figure 1 we can see that there are many more modules available in npm than in other package managers from other languages [23].

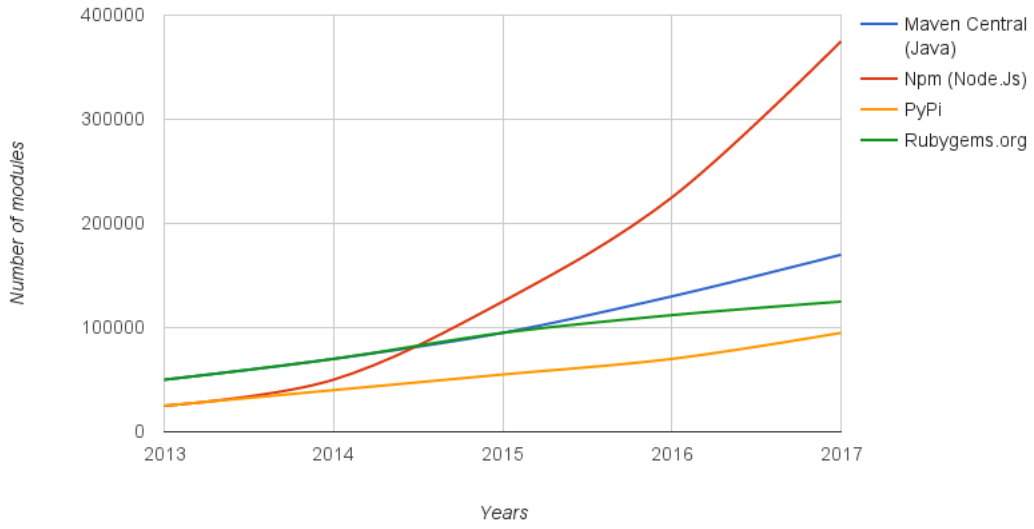


Figure 1: Number of modules available by different package managers through the years.

There are some drawbacks using Node.js. The fact that it is an interpreted language, intensive computational tasks may take much time and block the execution of the application. To resolve this issue, most of the times, heavy CPU tasks are offloaded to a micro service developed in another language.

2.2.2 LoopBack

LoopBack [1] is an open-source framework that runs on Node.js. It is developed by StrongLoop that was acquired in 2015 by IBM [24].

This framework allows the fast development of Application program interfaces (APIs). Behind that API is also an Object Relational Mapping (ORM) framework. It can be configured to connect to diverse data sources (i.e. databases, files, other APIs) using connectors. It then maps those sources into JavaScript objects and generates an API that handles all the data.

LoopBack comes with a set of base models that can be used and extended to simplify the development of the API. For example, user authentication can be achieved by extending the base user model. This way, LoopBack automatically handles all authentication.

Controlling the authorization to the API endpoints is also available through the use of Access Control Lists [25]. This way, we can specify that a certain role is allowed or denied to do some type of action (read, write, custom function) in the API. Roles can be easily assigned to a user when authentication is performed or when the user tries to access some data.

All configurations can be changed by editing JSON files. All aspects of the API are configurable. If there is the need to create a custom function for the API, we can add a JavaScript function to the model we want.

Since LoopBack is built on top of the Express framework [26] , it can access its features. Middleware is one of those features [27]. Middleware is a intermediate step that can be executed between other two steps. Those can be middleware too. For example, when a user tries to insert some data, an intermediate step can be called before the insertion to verify if the data is valid.

In Figure 2 we can see the general architecture of a LoopBack API [1]. All the red dots are where we can intercept data and add new features. When a request is made to the API, it first enters the HTTP server, then the API checks the authorization through ACLs. Then, the data is validated using the validations defined for each model, and finally the operation is performed recurring to the datasource (typically a database). Data is always passed as JSON objects.

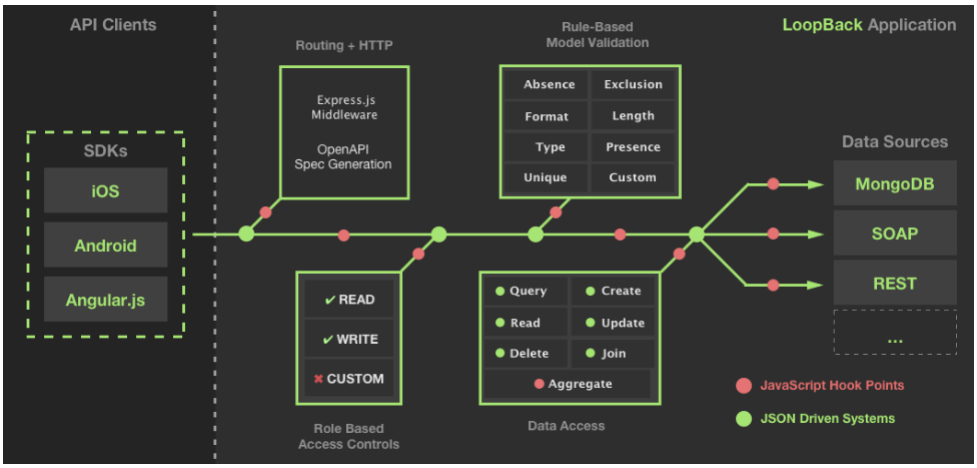


Figure 2: LoopBack data flow [1]

Because the framework is new and is not broadly used, the documentation is poor and sometimes non existent. It may be necessary sometimes to dive into the source code to discover some undocumented functionality.

2.2.3 AngularJS

AngularJS [17] is a JavaScript framework intended to be used for the development of frontend single page applications. It extends the functionality of HTML by providing new tags and attributes.

This framework introduces the following features:

- **Module** - Modules can be used to delimit all the components inside an application. For example, a global application can be divided in smaller ones, being each one a Module. Each controller has various Controllers and Views.

- **View** - Views are generally HTML pages with the presentation of the data passed by the controller.
- **Controller** - Controllers make the bridge between Services and the Views. It is used for small component logic.
- **Service** - A Service can be used by various Controllers, and is intended to carry the business logic of the client side application. It should handle the communication with external sources such as APIs. All the global application data should be stores in Services.
- **Directive** - Directives are attributes or tags that when applied to HTML code, extend its functionality. AngularJS already provides many directives (ng-repeat, ng-bind, ng-click...) but new ones can be created using simple JavaScript code.

When developing Single Page Applications (SPA), it is important to ensure that when a new section of the website is loaded, there is no refresh. That would break the concept of a SPA. With this in mind, Angular provides a special functionality that enables routing in the application without refreshing each time the user clicks an URL. We can define for each URL, what view it will redirect and what controller will be responsible for that view. The URL will take the form of *https://website.com//section*. The / can be removed using the HTML5 mode with the routing functionality. But in order for this to work, the browser has to support HTML5 history [28] and rules have to be added to the web server to redirect all URLs to the index.html.

One drawback in using this framework is that it is not search engine optimized. That means that certain search engine bots will not be able to retrieve content from all the pages since they are rendered in the client-side using JavaScript. But since our application will be private, and to view any part of the website the user must be authenticated, this issue will not impact the development.

Being one of the most used frameworks, brings the advantage of the large amount of libraries available to extend AngularJS functionality. And also, because it is developed by Google, the documentation is very rich.

2.2.4 LoopBack + AngularJS SDK

By using LoopBack and AngularJS together, we can take advantage of one feature that LoopBack offers: the **AngularJS SDK**. That means we can abstract the concept of the API on the client side and access it by calling simple SDK functions. We do not need to directly make HTTP requests to the API. Instead, the SDK handles that for us.

The SDK is simply a set of AngularJS services that are automatically generated by LoopBack and include call functions to the API. Data in those calls is passed as JSON objects.

One of the features that the SDK handles for us is the authentication management. API endpoints are most of the times protected and need users who access it to be authenticated. When a user signs in using the application made with AngularJS and the SDK, it will create an access token that will identify the session created. The SDK will store that access token and will send it in every request made to the API. The access token can

contain various information: user id, user name, session expiration time and any custom data we need in our application.

2.3 Comparison of front-end frameworks

The visual appearance and usability of an application is becoming more and more important every day. Users want to feel that their actions have instant feedback and their experience is somehow visually attracting. Along with those design aspects, it is also required that visual interfaces support today's use of various screen sizes. Today, we have a variety of devices that are used to access web applications: computers, smartphones, tablets, smart TVs and even VR headsets. All those devices vary in size, aspect ratio, dots per inch (dpi). So we must make sure that when developing a web application it will be adjustable to those different characteristics.

To help us develop a modern interface for the web application we will be using a visual front-end framework. Usually those frameworks make use of CSS and JavaScript. In this section we will compare three modern frameworks and decide the one that better suits our needs.

First, we have to present the concept of pre-processed styles. CSS can be coded directly using its own syntax. But that syntax most of the times can be very verbose and thus time-consuming. So, to overcome some difficulties, CSS styles can be written in either **Less** [29] or **Sass** [30]. These are two different pre-processed languages for CSS that enable the use of variables, functions and a better code organization. Less and Sass code are processed when building the application using their respective pre-processor. At this time, they are transformed in vanilla CSS code.

2.3.1 Bootstrap

Bootstrap [31] was first released in 2011 and is one of the first front-end frameworks released. It was originally developed by Twitter and is the most popular front-end framework.

It has various pre built components that can be used simply by assigning one of the defined Bootstrap classes to an element. Components include alerts, dropdowns, navigation bars, icon buttons, etc. Themes can be created for Bootstrap, overriding the default theme.

Bootstrap is responsive. It adapts the visual to different screen sizes. If a screen is small, elements are scaled down to fit the screen or even get hidden. The way elements scale or hide in different resolutions, is configurable by assigning classes to the intended element.

It supports both Sass and Less pre compilers in order to override default styles. It may be useful to change some of the code in order to get more performance.

Bootstrap also has JavaScript elements that provide extended functionality. JavaScript is not required for base Boostrap elements to work, but is recommended.

The documentation for this framework is very rich and has lots of examples.

2.3.2 Foundation

Foundation [32] is an open-source framework developed by a company called ZURB. It can be used both for styling websites and emails. This framework is also responsive and has various components that can be imported separately to minimize download times. Its documentation is very good.

2.3.3 Materialize

Materialize [33] is based on Google's Material Design [34]. The focus of Materialize is the user experience. As such, it gives the user feedback for every action in a way that the user feels connected with the application. Material Design is used by Google on the Android devices. Because of that, users may feel more comfortable with it

Like the other frameworks, it is also responsive and has various components that can be used right away with minimal effort. It is developed using Sass pre compiler but there are tools to enable the use of Less to change the source.

Changes can be made to the appearance overriding the default theme variables.

Feature	Bootstrap	Foundation	Materialize
Open-source	Yes	Yes	Yes
Responsive	Yes	Yes	Yes
Prepocessors	Sass, Less	Sass	Sass
Themable	Yes	Yes	Yes
Grid system	Yes	Yes	Yes
Import separate components	Yes	Yes	No

Table 1: Front-end frameworks comparison

As we can see in Table 1, all three frameworks offer a similar set of features. We chose Materialize because of its simplicity and modern appearance. Having a mobile like visual is a plus because users can access the platform in any device and have a seamless experience. The fact that it does not support the import of separate components will not be a problem. Most of the components will be used therefore there is no need for separate imports.

2.4 Web Communication

There is a need for web notifications on every application that deals with important information. When a message is received or a pending action is resolved, users have to be

notified about that. So in this section we will present and discuss technologies to enable real-time communication within a web application.

2.4.1 WebSocket

WebSocket [35] is a protocol that enables the real-time communication between an web application running in a browser and a server. This protocol uses the HTTP protocol to initiate the connection between the client and the server. This is known as the **Handshake**. If the handshake is completed successfully, the connection is upgraded to the WebSocket protocol over the TCP transport layer and functions has a full-duplex channel.

To connect to a server using WebSockets, it is used an URI like:

```
"ws:" "//" host [ ":" port ] path [ "?" query ]
```

The use of *ws* means the connection is not encrypted. To use an encrypted connection, *wss* must be used.

After the handshake, both server and client can communicate with each other at any time. This solves the problem of older technologies that used polling to the server, where the client had to send requests every x seconds to the server to check for updates.

Because not all browsers support the use of WebSockets, we will next show a framework that deals with the connection establishment and checks for the best method for client-server communication.

2.4.2 Socket.io

Since we are using Node.js for the development of the platform, we can take advantage of the modules that exist for it. **Socket.io** [36] is a server and client library that implements real-time connections. It can run on any browser (desktop or mobile).

The connection between the client and server is established with XMLHttpRequests (XHR). Then, if the connection is successful, it tries to upgrade the connection to use WebSockets. If WebSockets cannot be used it stays with XHR. This way, there is no need to implement web sockets from scratch because protocol establishment and handling is already implemented by the library and compatibility with older browsers is also handled.

2.5 Comparison of web servers

We have presented, discussed and selected the frameworks that we will be using. Now it is appropriate to decide what software will be used to serve the web application.

Although Node.js can act as a web server using the HTTP module [37], there may be the need for load balancing or reverse proxy. To ensure a robust way of hosting and serving the web application it may be more suitable to choose an HTTP server.

The servers we will compare are **Apache HTTP Server** [38] and **NGINX** [39]. Both of them are open source and the most used globally [3]. We can see that in Table 2.

Name	Top 10 000 sites	Top 100 000 sites	Top million sites	Entire Web
nginx	3,877	35,538	298,351	35,826,971
Apache	3,697	38,824	374,559	69,775,578
IIS	1,848	18,533	123,618	58,881,364
Varnish	1,319	5,949	25,816	2,509,979

Green means that the usage has increased in the period of 27/08/2017 - 3/08/2017. Red means the usage has decreased in that period.

Table 2: Web server usage statistics of the top 4 web servers [3].

In terms of features they offer, we analyzed both solutions and observed that they can perform the most usual tasks, so both of them are suitable for delivering our web application.

Because both servers have the most used features, the key point for choosing them is the performance. The huge difference between Apache HTTP Server and NGINX is that the former uses threads for each client and the later uses an asynchronous event-driven approach. That means that on NGINX, all connections are handled in the same thread. When there is activity in a connection, that action is scheduled into a FIFO queue and processed as soon as other actions are dispatched. This way NGINX does not suffer the problem of spawning many threads and overflowing the CPU with thread switching [40].

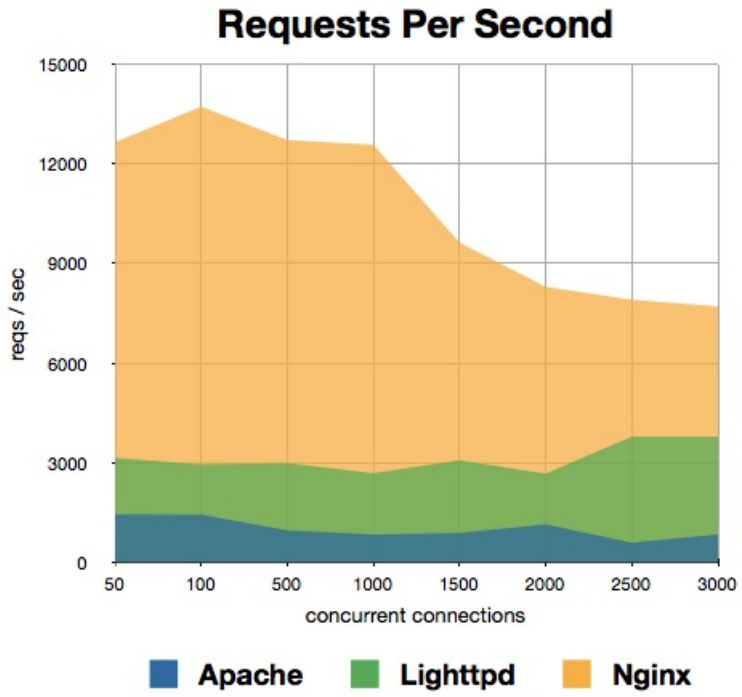


Figure 3: Requests per second by number of concurrent users [2].

We can see in the graph of the Figure 3 that NGINX clearly outperforms Apache HTTP Server [2]. This is the reason why we will use NGINX as our web server and will try to make extensive use of its features.

2.6 Node.js Process Managers

Because we are going to use Node.js, we feel that we must use a process manager. A process manager is a piece of software that will be responsible for running and managing or server application when on a production environment. If the application crashes, the process manager makes sure that it will restart again automatically and if we want to deploy a new version of the application it will make sure that at least one instance of the application is running to prevent disruption of service.

We analyzed and tried three process managers to decide the one to use in the deployment of our application. All process managers run on top of Node.js and are the following:

- **Modemon [41]** - This is a simple manager that auto reloads applications when it detects file changes or the application crashes.
- **StrongLoop Process Manager [42]** - It is a process manager developed by StrongLoop, the same developers as LoopBack. It is best suitable to use together woth applications made with the LoopBack framework. As a lot of features such as application monitoring, metrics capturing, cluster support (running various instances of the same application), load balancing.
- **PM2 [43]** - PM2 is developed by Keymetrics.io and is intended for generic usage. It can run applications outside the Node.js environment, like python for example.

it also has a vast number of features like the StrongLoop Process Manager.

Features	Nodemon	StrongLoop Process Manager	PM2
Auto reload application	Yes	Yes	Yes
Cluster mode	No	Yes	Yes
Scale number of nodes in a cluster	-	Yes	Yes
Load balancing in a cluster	-	Configurable	Default only
Saves metrics	No	Yes	Yes
Saves logs	No	Yes	Yes
Integrates with external monitoring services	No	DataDog, Graphite, Splunk	Keymetrics.io only

Table 3: Node.js process managers comparison

We can see in Table 2.6 that LoopBack offers the most features, there are many other features that are not mentioned here for sake of simplicity. Nodemon clearly does not suit our needs, due to lack of features, it is intended for development only.

Upon trying these process managers we felt that PM2 is the best choice to our needs. Of course LoopBack Process manager provides us with more features, but it is harder to configure than PM2. Furthermore, PM2 documentation is significantly better and we think it has the exact features we need.

Considering the Nodemon's ease of use, we will use it for our development environment and will use PM2 for our production environment.

3 Planning

This chapter presents the planning of all the work performed for this project. The project was divided into two semesters of work. On the first semester we studied the technologies and the old platform, planned the new platform and its architecture. We also developed a small prototype to help increase our knowledge about the technologies. The second semester was mostly dedicated to the implementation and validation of the proposed platform.

3.1 Tasks performed in the first part of the project

As said before, the first semester served as the foundation for the work performed on the second semester.

There was not a rigid plan for this first part. We just had monthly sessions where we defined the work for the next month. We can see the Gantt chart of the work carried in the first semester in Figure 4.

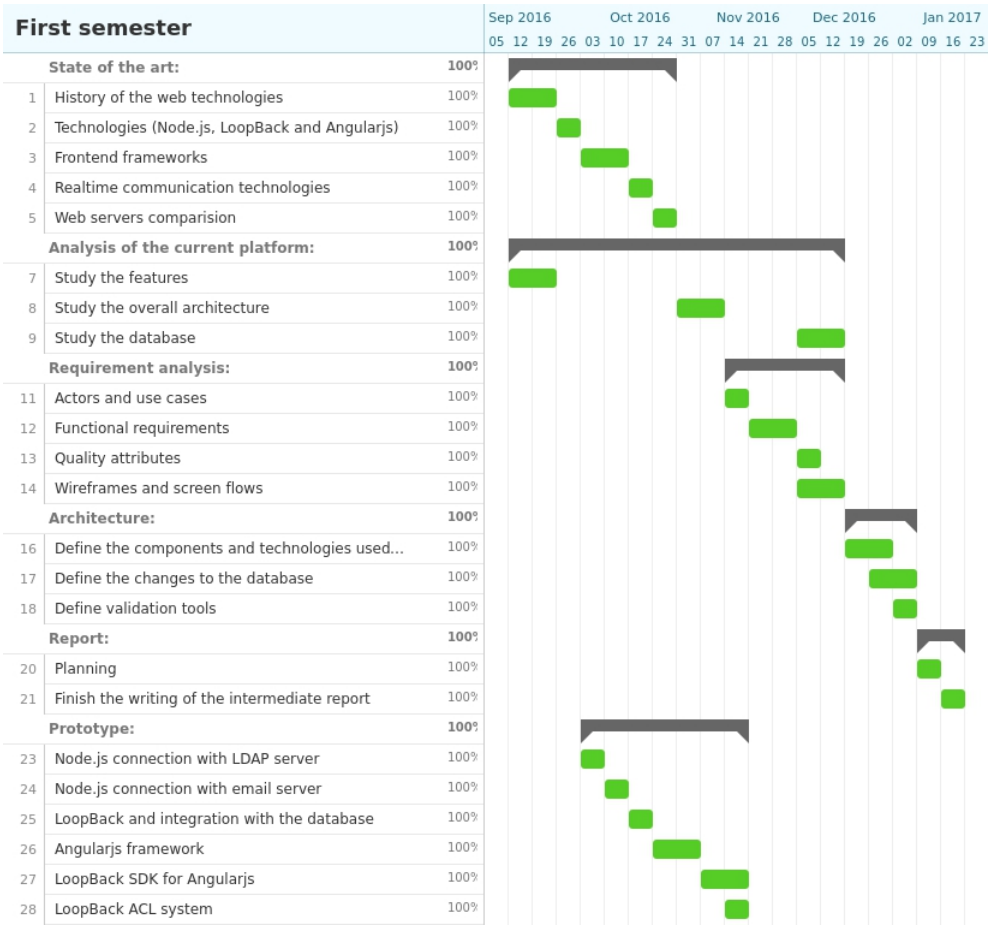


Figure 4: Schedule of the tasks performed in the first part of the project

It should be noted that the developed prototype was of great importance for the success of the project. It helped us to have a better knowledge of the technologies and also showed

what problems we would face when developing the real platform. We learned a lot from this prototype and think it was essential for the development of the final platform.

3.2 Life cycle

There are some aspects to take into consideration for choosing the proper life cycle:

- The scope of our project was well defined and we had solid requirements.
- We were transitioning from an old platform to a new one.
- Presenting the platform to a small set of users first, can help us to improve the platform and have a better feedback. Thus, delivering features as they are implemented will ensure users can use them as soon as possible.

These aspects made us think that using the Waterfall model with Staged Deliveries was the best way to accomplish our goals. Because requirements were well defined, we estimated the time required to implement and test them. And we also defined delivery dates as when a major set of requirements would be implemented. Having delivery stages throughout our project is important because this way we will be releasing the stable version to the public and receive important feedback. This feedback can be used to improve the user experience.

3.3 Development plan

We defined our development plan based on the MoSCoW prioritization of each requirement and the relevance of the requirements. For example, a feature that does not exist in the old backoffice is given a high priority and will be developed before the features already in production. Another aspect is the requirements that are tangent to all the platform (e.g. role management, log system). Tangent requirements must be developed before all others, because they will be required by the rest of the application.

First, we performed the effort estimations and then we combined them with priorities to get a schedule of the work of the second half of the project.

3.3.1 Effort estimation

Estimating the effort required for this project was a very important step to take. This ensured that we planed an optimal schedule for the implementation process.

We chose to use a **bottom up** strategy to estimate time required where the implementation process is subdivided into smaller tasks. Because completed requirements are what define the final product, we use them as the smaller tasks and estimate the time required for each of them. Implementation of the requirements already **includes the testing time**.

We produced an average estimate for each task by estimating the effort for best and worst case scenarios. To have a more realistic view of the time required, we added a slack

variable to the time. That increases the estimated time for each task and ensures that the scheduling of tasks to be more realistic.

Because we had already developed a prototype using the required technologies, we had a better insight of the times required for each task.

The result of the estimations can be seen in the next section, where we present the schedule for the implementation part of this project.

3.3.2 Schedule

In Figure 5 we present the schedule for the second part of the project in a form of a Gantt chart. This is an overview of the scheduled plan.

The project delivery stages can be marked as the following:

- **21/03/2017** - The card access management can be used by the system administrators.
- **13/04/2017** - The dissertations are operable and can be used by students and professors.
- **25/04/2017** - Information features are available and can be used by any user.
- **5/05/2017** - Management of the information is available.
- **19/05/2017** - All features are implemented.
- **2/06/2017** - Release with visual optimizations and final adjustments.

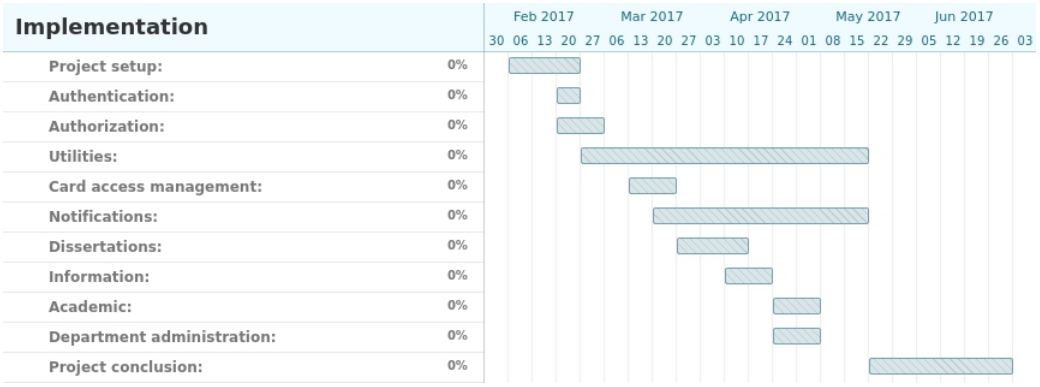


Figure 5: Overview schedule of the tasks to be done on the second part of the project

3.3.3 Work done in the second half

In Figure 6 we show the actual work done on the second half of this project. We can clearly see that there was a 2 week delay in the final part of the implementation process. As in all software development projects, the estimations were not 100% correct, but nevertheless we could manage to implement all the features planned. To note that there was a change in the requirements, where the Courses feature was replaced by the Key Request feature. The Courses feature remains as requirement but with a low prioritization.

Since the first deployment, we started to gather information from the users through meetings that occurred along the semester involving Streamline, directors and professors. There were 3 main meetings along the process, but we were always in contact to ensure the project was in its right way.

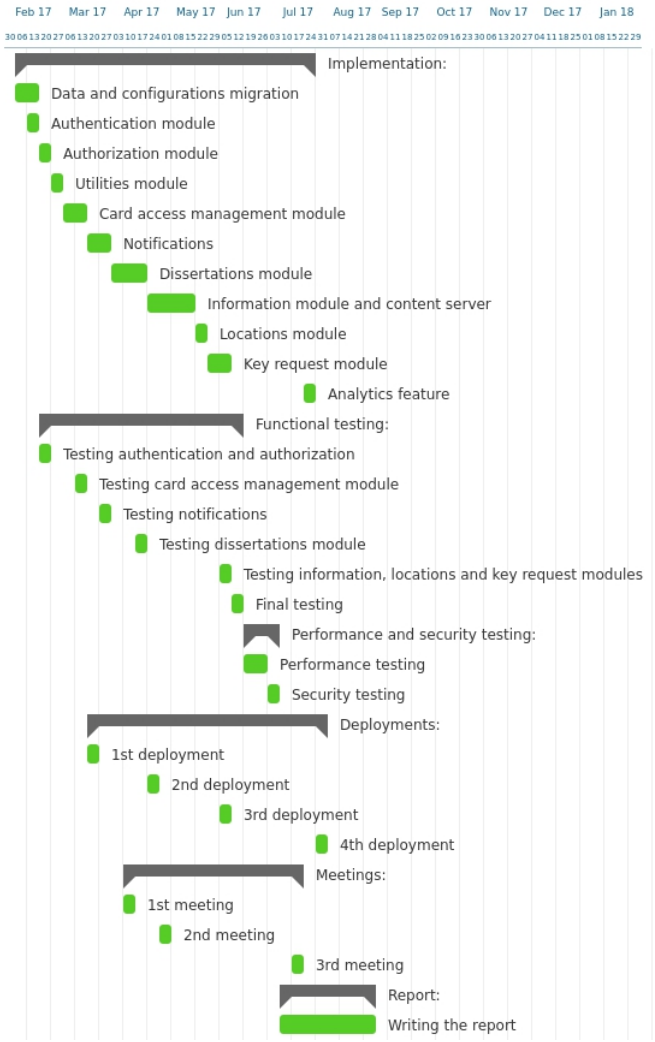


Figure 6: Tasks performed on the second part of the project

3.4 Risk management

In a project, it is best practice to identify the risks that can occur during their lifetime. After identifying them, a good mitigation plan must be produced.

For this project, a risk management assessment was made, and is presented in the following tables. We classify risks according to their likelihood and impact. Likelihood and impact are classified as Low, Medium or high. Then, a brief mitigation plan is presented.

Risk	Unfamiliarity with the technologies used
Description	Unfamiliarity with the technologies used can lead to delays in the development process
Likelihood	Medium
Impact	Medium
Mitigation plan	This risk was mitigated by developing the prototype on the first half of the project

Table 4: Risk - Unfamiliarity with the technologies used

Risk	Migration of the database
Description	When migrating the data from the database, that can lead to data inconsistency
Likelihood	Medium
Impact	Medium
Mitigation plan	This risk was mitigated by testing the migration scripts using a copy of the real database and having a backup if anything would go wrong

Table 5: Risk - Migraion of the database

Risk	Integration with third party database
Description	Integration with the the card management database can lead to inconsistency and errors
Likelihood	Medium
Impact	High
Mitigation plan	We mitigated this risk by comparing the results of our solution with the results from the third party software

Table 6: Risk - Integration with third party database

4 Current platform

The old backoffice (DEEC Backoffice) has been in production for nearly 10 years now. It was partially developed by **Streamline** and is the main platform used to view/manage dissertations, view academic information, manage personal data, view and edit other types of information related to the department. It is used by everyone in the department, including students, professors, secretary and administration. In Figure 7 we can see the login page of the platform. To note that in order to access the rest of the website, it is necessary to be authenticated using a valid email and password. The authorized emails are from the domain **@alunos.deec.uc.pt** for the students and **@deec.uc.pt** for all other people.

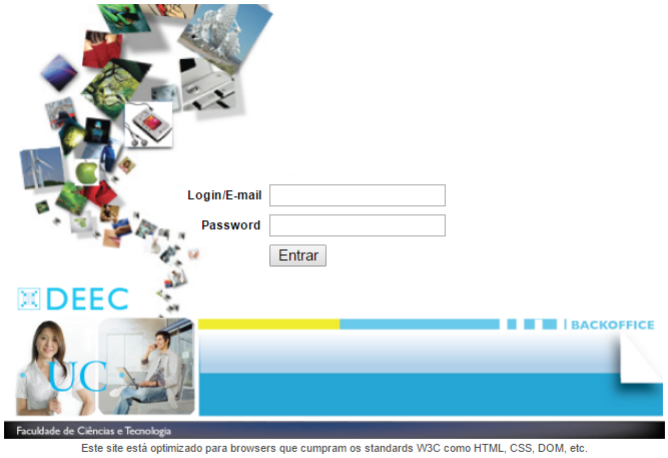


Figure 7: Current login page of the DEEC Backoffice

Next we will study the platform we are going to migrate and the technologies being used by it. Also, we will explain the architecture of the old system and how the components interact between each other.

4.1 Current features

Currently, the following main features are provided:

- **Academic management** - Course information and its related materials that are available to the students. Includes also dissertation management (insertion, jury assignment). Students can apply to a dissertation and professors can accept them.
- **Documentation archive** - People from the department's administration can view old documents.
- **Personal data** - Possibility to view and edit personal data by the administration.
- **Building** - Information to all the people about the places that exist in the department.
- **Communication** - Administration can add news about the department.

- **System administration** - Several utilities used to perform tasks by the system administrators (Streamline). Diverse software configurations can also be seen here.
- **Inventory** - Insertion and listing of inventory items. Reports about the inventory can be consulted here.
- **Contacts** - Information about the most relevant areas of the department. Including people contacts of the different areas. Can be viewed by all people.

Most of the features are deprecated or not being used anymore. Such is the case of the courses material. Students in the department use the platform **Inforestudante** [44] that is used across all University of Coimbra, so there is no need to provide materials in two separate platforms. Another feature not being used in the platform is the insertion of news. This has been moved to a new service developed by Streamline named **CorporateTV**. Other functions such as the document viewing will be moved to a cloud file system. We have few features that are actually being used by the users: dissertation management, building information, contacts, personal data. Although being used, they do not match the actual needs, thus they need to be redefined.

4.2 Overall architecture

As we can see in Figure 8, the current platform runs on a virtual machine running CentOS [45]. That machine runs **Apache HTTP Server** and has about 30 virtual hosts. Most of those hosts are small websites that have very few requests per day.

The DEEC Backoffice is developed in PHP (without the use of any frameworks) and is totally served by this machine.

There are 3 more services that are accessed by the Backoffice to support its operation. They are:

- **DEEC Database** - The database that contains most of the data present on the Backoffice. It runs MySQL 5 [46]. We show the most important tables below in the database structure section.
- **DEEC Email Server** - This service is used by the Backoffice to send notification emails about important actions that occur on the platform.
- **DEEC LDAP** - Lightweight Directory Access Protocol (LDAP) stores all credentials from all the accounts. It is used for authorization. When a login in the Backoffice occurs, it checks on the LDAP server if the password is associated with the email and the account is valid.

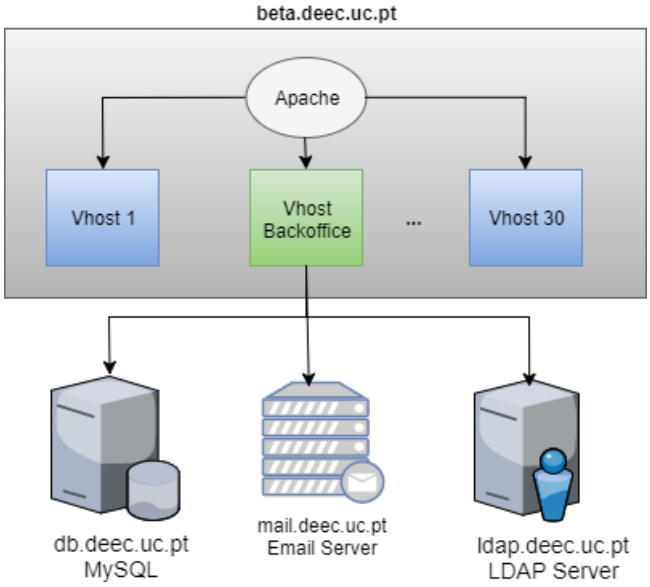


Figure 8: Current architecture used by the DEEC Backoffice.

Currently, because all pages are rendered by PHP and few JavaScript is used client side, there is no much interactivity. Users have to wait for the full page reload each time they want to perform an action. With the power that user devices today offer, we can take advantage to provide a more interactive user experience. So, we have to make sure that we combine the power of the servers with the power of the clients to provide a good experience to the users.

With regards to the business logic, the Backoffice does not provide an API. This means that if for example a mobile application needs to be built, all logic has to be built from scratch, and that would mean that there would not exist a central logic. By providing a central API that can be used by several applications, the development time of new platforms and tools can be drastically decreased. This is one of the reasons the building of a new system with an API is so important.

4.3 Database structure

As said before, all the DEEC Backoffice data is stored in a MySQL database. It currently has **125 tables**. We can view all the tables in Figure 9. We will have to deal with this amount of tables to maintain compatibility and consistency between the new and the old platform.

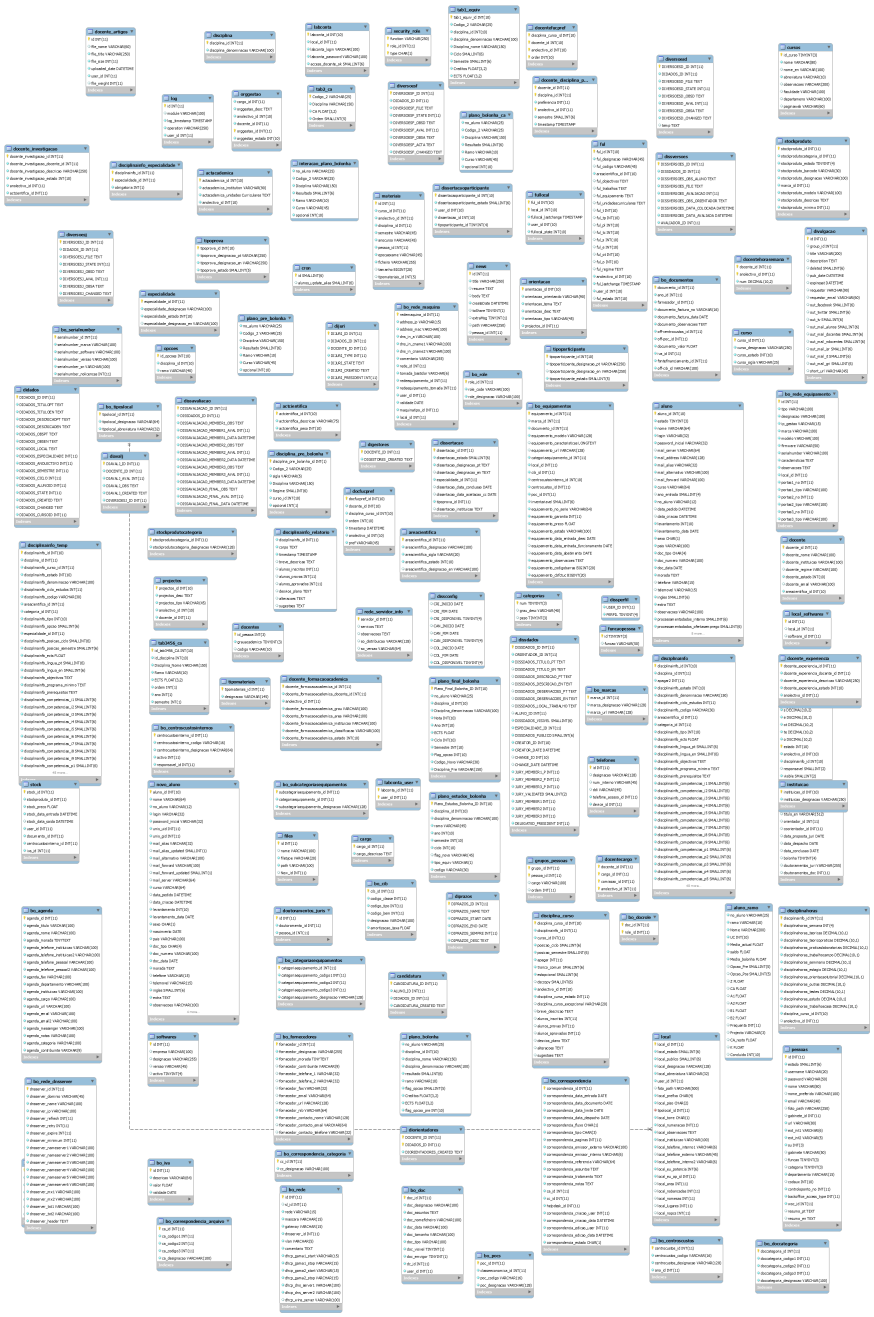


Figure 9: Old database Entity Relationship (ER diagram.

The problem with this data model is that many of the tables are not used and some of them have no data stored. Some tables have the same purpose because they belong to different versions of the old backoffice. We will have to conduct some migrations changes to support the new backoffice (more on this later).

Table	Description
aluno	Contains all the students of the department
pessoas	Contains all the non-student persons of the department
especialidade	Contains the course specializations
didados	Contains all the dissertations
diorientadores	Maps a person who is a professor to a dissertation
local	Contains the several places of the department and their information
bo_tiposlocal	Contains the several types of places
aluno_ramo	Maps a student to a specialization. Contains information about the student in that specialization.
disciplinainfo	Contains the information about all the courses
bo_doc	Contains names and paths to several administrative documents

Table 7: Important database tables

From all the tables, 45 are not being used and **10** of them do not even contain any data. Analysing dates of insertion in other tables gave us a notion of features that have not been used a long time ago like for example the insertion of course materials.

This database schema has a significant problem: **it does not have relations between tables and no primary keys in most of the tables**. The relations are managed by the PHP code. So, when a query is performed on the MySQL database it will not take advantage of the performance it would have if records where indexed.

Giving the fact that the database has several tables that are not being used, some relations do not exist and the need for new tables there is a strong need for changing the database structure. In chapter 5 we describe the changes we will make in order to support the new features we intend to deliver.

4.4 Flaws of the old system

Apart from the problems already mentioned (deprecated technologies, unfulfilled needs, unstructured code, no separation between logic and presentation layers, no responsive

interface) we tested the software for vulnerabilities and errors.

By running simple tests, we verified that the system is vulnerable to code injection [47] both SQL Injection and Cross-Site Scripting [48]. We will describe these vulnerabilities later when defining quality attributes for the new platform. We identified the source for this problem being that data is not being properly escaped and methods for saving data to the database are deprecated.

5 Requirements analysis

Now that we explained the current backoffice, it is time to come up with the requirements for the new platform. We will be taking the current features and improve them, resulting in a updated set of requirements. Also, we have to consider the new needs that the DEEC has. Because of that, new requirements must be implemented.

5.1 New needs

Next, we will present the new features that must be implemented. Some of them are improvements of the current platform and others are result of the emergent necessities of the department.

Role management

Authorization of every function of the platform must be strictly controlled through roles. A role can have various rules that grant or deny access to a specific function. Currently, one person can have only one role and the only way of changing that is by editing the database. With the new system, we will have to ensure that users can have multiple roles and a way to edit them in the platform.

Card access management

The department has a new door access system. Every person in the department can have cards associated with them (cards use RFID [49] to be identified). Those cards enable their owners to access the several doors of the department according some rules. Cards, rules, doors and users can be managed using the software acquired when the system was installed. The problem with this software is that it has very poor performance ($10\tilde{2}0$ seconds delay between operations) and only works using Microsoft Windows [50] operating systems. We thought that there must be a better way of managing the cards, providing the card managers with a better client software. Upon a brief analysis of the software, we discovered it uses a database where all the information is stored. We will then, use that database and access it using the new backoffice to provide card managers with a better and faster interface that can be used anywhere and in any operating system. Of course the card software has many features, but most of them are not being used. As such, we will only implement a subset of those features in our platform.

Displaying of information

Currently, the displaying of information and contacts of the different areas of the department is very basic and not very well structured. We will have to ensure that a information browsing page where users can search and view relevant information is implemented. That page must be modern and organized.

Physical keys management

Currently, the key management is made without the aid of any software. When a person wants to request a key, they must fill a request paper and give it to the administration that will decide if the request is authorized or not. Then, if the request is accepted, the person

who requested must go to the key holders show the signed authorized request and get the key. This process is represented as a Business Process Modeling and Notation (BPMN) diagram in Figure 10 (authorization process) and Figure 11 (lend/return process).

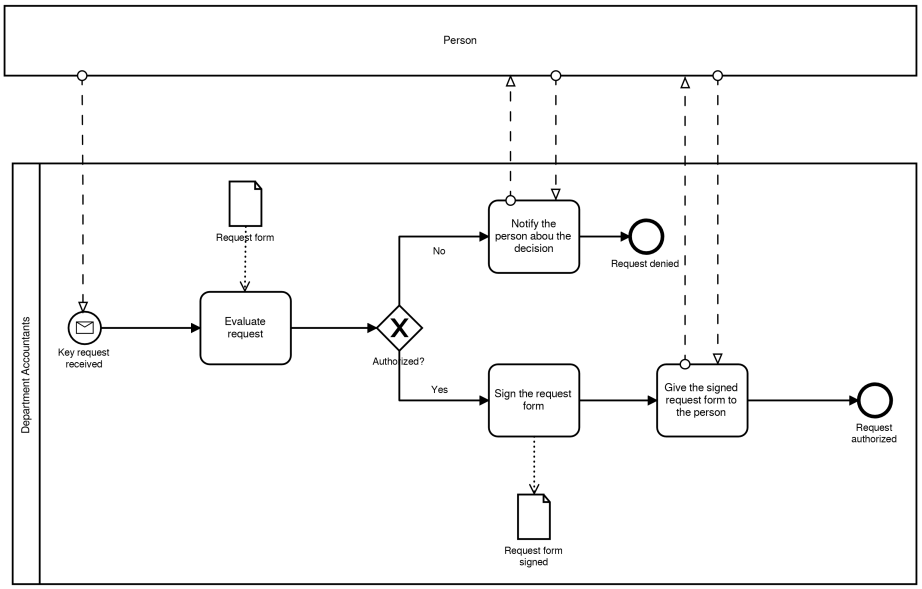


Figure 10: Key request authorization process

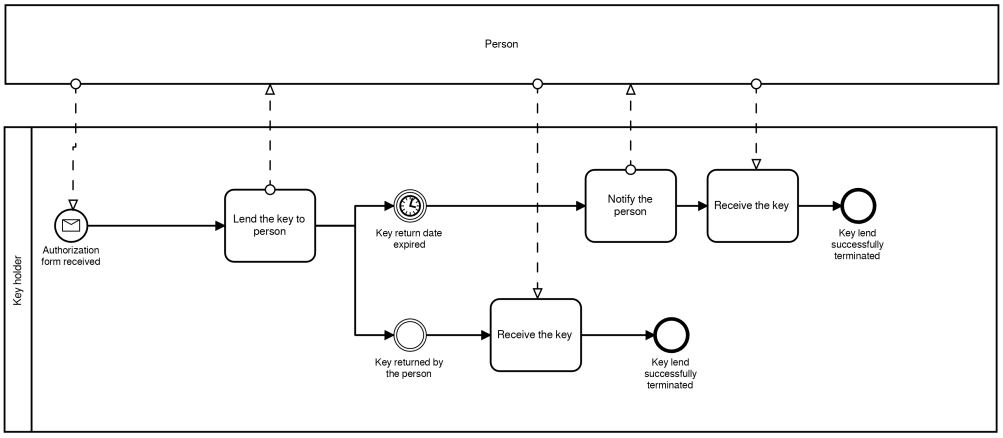


Figure 11: Key request lending process

Our objective is to include this process in our platform, allowing any authenticated user to request a key, key managers to authorize/deny those requests and to mark them as lent/returned. If key are overdue, notifications must be shown to alert users that they must return the key. The system also must be able to show who has a certain key, occupied keys and free keys.

Logging system

In every system there is a need to store important information about actions that occur. The platform we are going to develop is no exception. In order to help system administrators to better know what is happening, we need to implement a simple logging system

that allows simple configurations such as the level of log detail. The integration of this logging system with the rest of the application must be easy.

5.2 System Actors

Actors will interact with the system through the web application. The actors we present here are going to be (most of the times) the roles of our Role Management feature. Those actors are presented in Table 5.2.

Actor	Description
Unauthenticated user	Any user that is not authenticated.
Authenticated user	A user that is authenticated with a DEEC email and password.
Student	Any student of the department that is authenticated.
Professor	Any professor of the department that is authenticated.
Dissertation manager	Any person (excluding students) that is responsible for managing dissertations and critical information about them.
Secretary	Any person beloging to the secretary functions.
Key managers	This includes persons who have authoritative impact on the key requests and the key holders.
System administrator	People that are system administrators. Typically people from Streamline.

Table 8: System actors

Through the use of the Role Management feature, more roles can be added. So these actors will change with the evolution of the platform. They will be modified throughout the platform lifetime.

All the actors presented, except the unauthenticated user, must be authenticated. That means that every student, professor, dissertation manager, secretary and system administrator are able to perform the actions designated to the actor authenticated user. This will simplify the presentation of use cases because there are many features that all authenticated users can perform.

5.3 Use Cases

We decided to subdivide use cases in 9 types. This will make it easier to present the information in a more organized way.

- **Authentication** - Use cases related to the authentication in the system.
- **Authorization** - Use cases related to the management of the several user roles. This includes the association or disassociation of a person to a role.
- **Dissertations** - Use cases related to dissertation viewing, appliance, insertion and overall management.
- **Card access management** - Use cases related to the insertion of access roles, association of cards to persons, role permissions management.
- **Academic** - Use cases related to academic purposes (viewing details of the courses, edit them).
- **Information** - Use cases related to the showing and edition of useful information to the users.
- **Information management** - Use cases related to the management of important information related to the department and its users.
- **Utilities** - Use cases related with the management of the system (most of the times used by the system administrators).
- **Key request** - Use cases related with the request, authorization and lending of physical keys.

The use cases we present here are separated for each actor. This simplifies their organization.

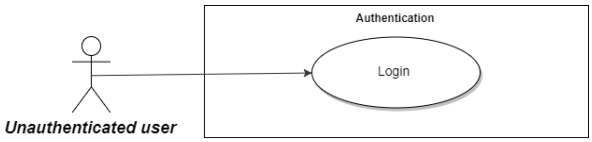


Figure 12: Use cases for the unauthenticated user

Figure 12 shows that the only task an unauthenticated user can perform is the login.

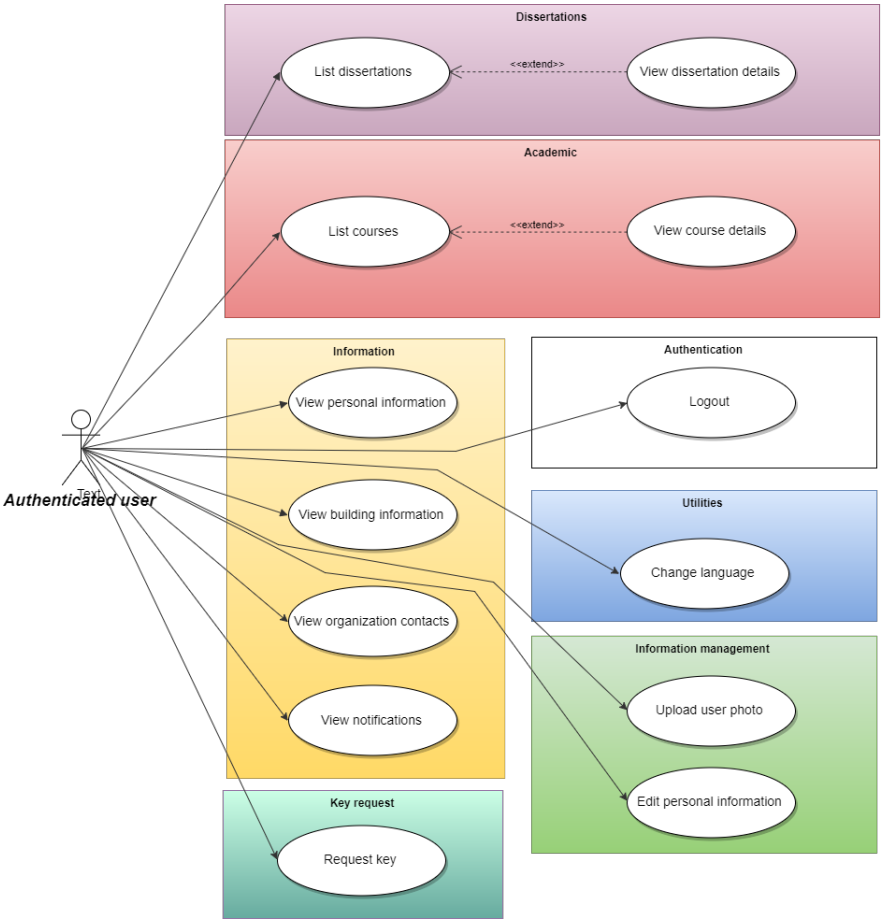


Figure 13: Use cases for the authenticated user

Every authenticated user can read all public information within the platform. And particular information to them such as notifications and personal information. Also can perform the logout. This is shown in Figure 13.

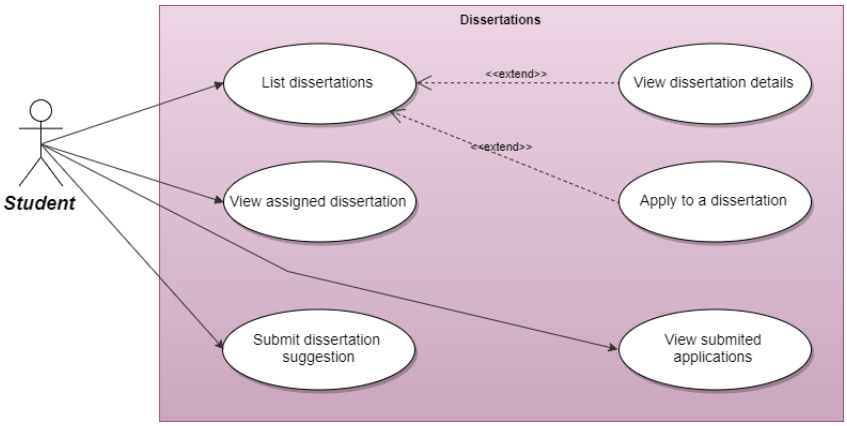


Figure 14: Use cases for the student

For now, students will only be allowed to apply to dissertations and view the dissertation

they are assigned to, as shown in Figure 14.

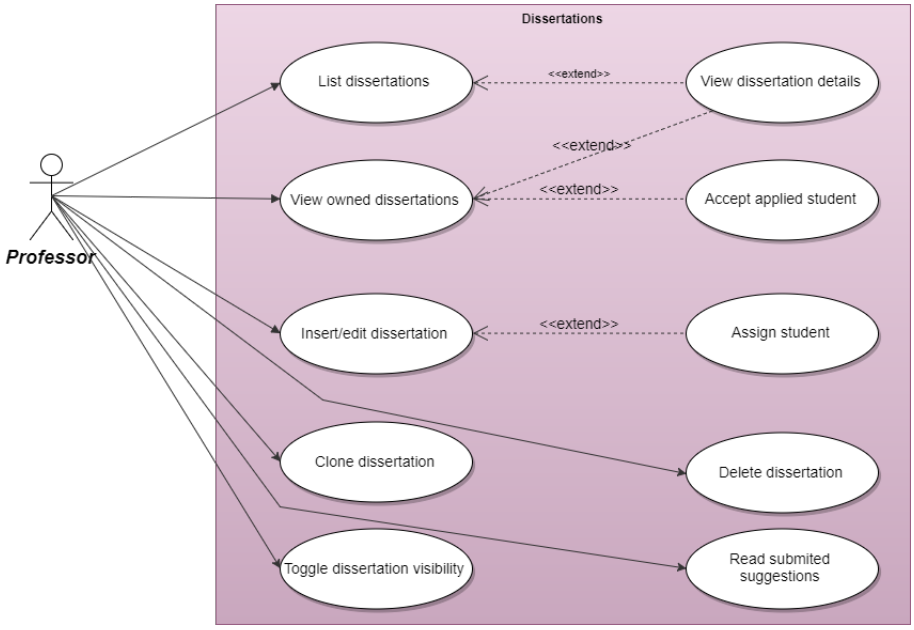


Figure 15: Use cases for the professor

As shown in Figure 15, professors will only have access to dissertation actions such as insertion and assigning students to them.

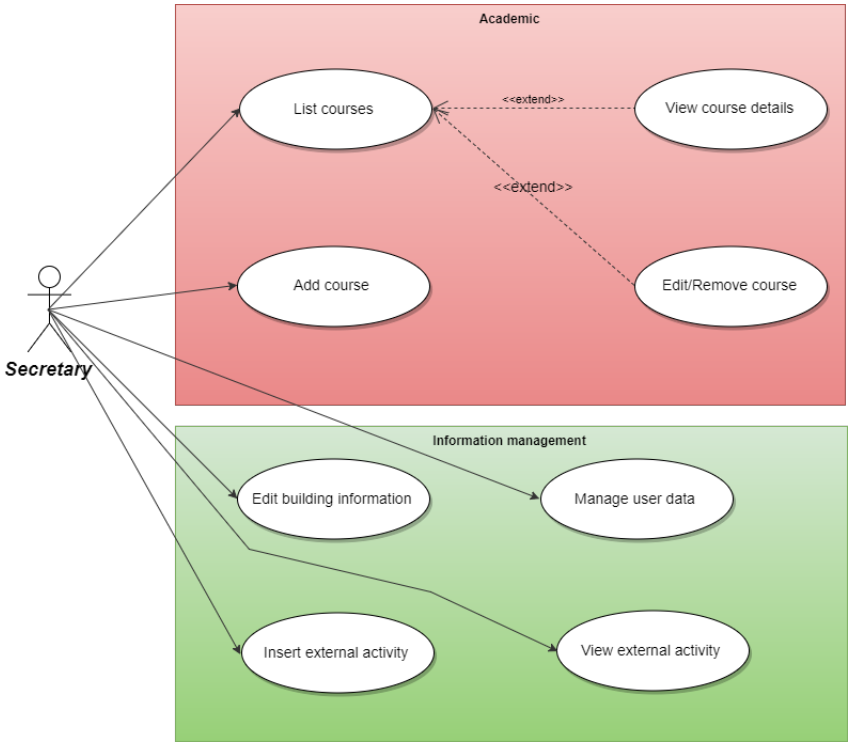


Figure 16: Use cases for the secretary

Secretaries can edit all the information about the department including user details and course information. This is shown in Figure 16.

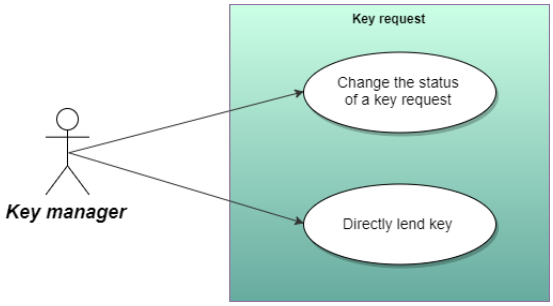


Figure 17: Use cases for the key manager

As shown in Figure 17, key managers are responsible for authorizing/denying key requests and to actually lend the keys.

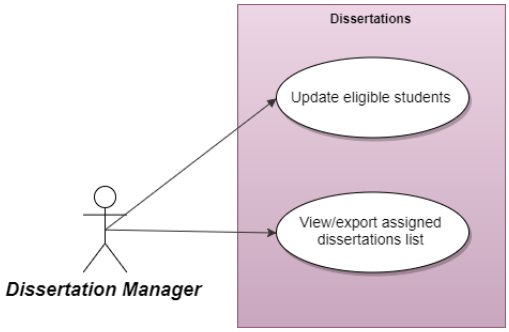


Figure 18: Use cases for the dissertation manager

As shown in Figure 18 dissertation managers assign juries to a dissertation and edit the students who are eligible to apply to a dissertation.

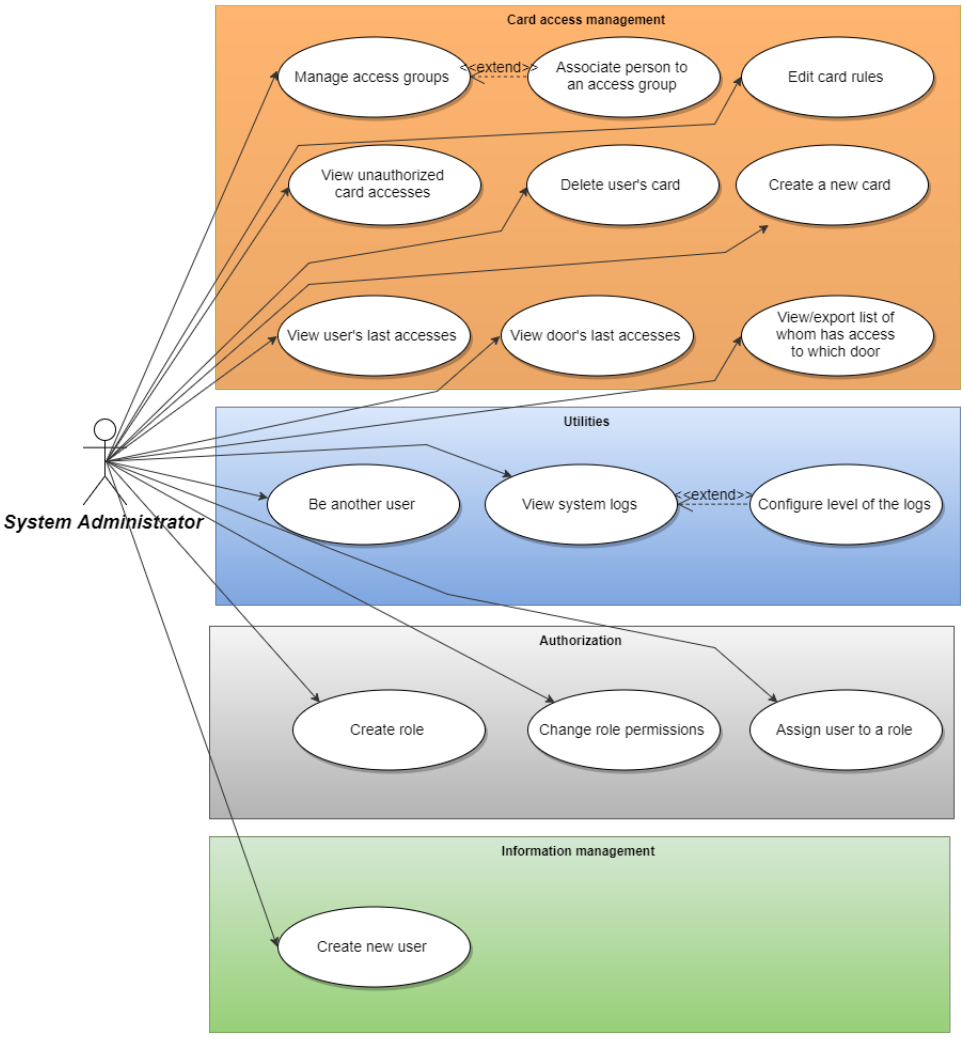


Figure 19: Use cases for the system administrator

System administrators manage the cards of the users, assign users to roles and have access

to some administration tools as shown in Figure 19.

5.4 Prioritization

Given the amount of requirements that the application will have, it was decided that we will be using a prioritization method. This will ensure that we focus first on the most important requirements and then we continue to the least important.

We opted for using the **MoSCoW method** [51] because it is very simple and enables us to separate requirements in four categories:

- **Must have** - Those requirements are the core of the application. If one of them is not correctly implemented, the project is not successful.
- **Should have** - Those requirements are also important and should be included in the application. But they are not as important as the core ones.
- **Could have** - These requirements are not so important but they would improve the application.
- **Won't have** - These requirements are not important and are not included in the planning of the project (material for future work).

For labeling each requirement, we analyzed the most used features of the current platform. The most used ones will be the first to be implemented and the least used will be the last to be implemented.

The use of the Won't have category will make the requirement be included in the platform roadmap. It will be a feature of future work and will not be addressed on the implementation section of this document.

5.5 Functional requirements

After having all the use cases, we defined the requirements having the following attributes for each one:

- **ID** - Identification of the use case, used to identify it thorough the document.
- **Requirement title** - The title representing the goal of the use case
- **Primary actors** - Who will be the main user/group of users performing the action
- **Secondary actors** - Who will be affected by the action
- **Description** - Description about the requirement, can include special cases that can occur
- **Flow** - The flow of action. What the primary actors will follow when performing the action
- **Pre-conditions** - Conditions that have to be met for the action to happen

- **Post-conditions** - What is changed after the action takes place
- **Priority** - The priority of implementation has described before.
- **Specificity** - The level of detail of the defined requirement.
- **Source** - The category where the use case belongs to (is one of those 9 types we showed before)

Because we have so many requirements, we thought it would be better suitable to show them in Appendix A. This ensures that we don't have so many tables in the middle of the document.

5.6 Quality attributes

To make sure that users can perform the actions we have just presented, we must guarantee a certain degree of quality of the software. This quality is often measured based on the **quality attributes**. These attributes are composed of metrics that evaluate the software in terms of performance, security and many other keys in the a way a system works. We will now present the quality attributes for the new platform.

5.6.1 QA01 - Platform Availability

Being used by all the people of the department, our platform must guarantee a minimum availability. Users must be able to access it any time they need, so we have to have a service that is always running and is able to recover after a malfunction occurs. This availability will be measured in the components we will develop, we have no control over the whole department network so we have to architecture it with the means we have.

ID	01
Title	Platform Availability
Stimulus source	Requests to our server components
Stimulus	A component crashes due to some malfunction
Conditions	Component was running
System elements	Server components
System response	System continues its normal operation and a new instance of the component is started to substitue the faulty one
Metrics	Percentage of time the platform is up during a certain period of time; Time to restart server node after failure

Table 9: QA01 - Availability

5.6.2 QA02 - Platform Scalability

The second quality we must address is the fact that our platform must be able to support large amounts of user requests. At least, the system must be able to handle a minimum of **500** requests per second with less than **100ms delay** of response. We think this is enough for all the users in the department. But of course we want it to be able to scale if we expand it to other departments or other user types.

ID	02
Title	Platform Scalability
Stimulus source	Requests to our server components
Stimulus	Large amount of requests per second
Conditions	-
System elements	Server components
System response	The system is able to maintain a minimum response time and produce minimum errors
Metrics	Average response time (ms). Average errors per total requests (%)

Table 10: QA02 - Platform Scalability

5.6.3 QA03 - Security (Code Injection)

We will now address the security of our platform, starting by code injection. Code injection is one of the top vulnerabilities according to the OWASP TOP 10 Project [52]. This vulnerability enables malicious users to inject code in a page/API endpoint and execute that code on the server or other users machines. From that code execution can result the loss or modification of information, account hijacking (stealing user credentials) and many other things that compromise the normal functioning of the system. This is of course a major issue that must be addressed and we must make sure that it cannot happen in our entire platform.

We will address the main code injection vulnerabilities: Cross-Site Scripting (XSS) vulnerability [48] (injection of JavaScript code) and the SQL Injection [47] (execution of SQL queries on the database).

ID	03
Title	Security (Code Injection)
Stimulus source	Malicious user
Stimulus	Injection of malicious code
Conditions	-
System elements	Server and client components
System response	The platform escapes the code and prevents it from running in the server and web application
Metrics	Number of successful code injection attempts

Table 11: QA03 - Security (Code Injection)

5.6.4 QA04 - Security (Cross-Site Request Forgery)

Continuing with the security aspect of our platform, another vulnerability we must consider is the Cross-Site Request Forgery (CSRF) [53]. This vulnerability enables a malicious user to make other users perform actions on the platform without their knowledge or consent. The simplest case is when a malicious user posts online an image where the source is an URL to an action on the targeted platform. If the the platform is not protected, when a regular user tries to load that image it will load the actual URL and perform an action on the server. This issue must be addressed and we must make sure we protect against this type of attack.

ID	04
Title	Security (Cross Site Request Forgery)
Stimulus source	Malicious user
Stimulus	Request from an untrusted source
Conditions	-
System elements	Server API
System response	The API refuses to run that action and the action is logged
Metrics	Number of successful CSRF attacks

Table 12: QA04 - Security (Cross Site Request Forgery)

5.6.5 QA05 - Security (Broken Access Control And Session Management)

Now we enter in the authorization part of the application, the Broken Access Control And Session Management vulnerability[54] comes from a poor authentication mechanism with lack of encryption where credentials are passed and stored non-encrypted. We must insure that user credentials (like passwords and access tokens) must not be accessible by no one other than the user to whom they belong. We must also have a way to invalidate user sessions.

ID	05
Title	Security (Broken Access Control And Session Management)
Stimulus source	Malicious user
Stimulus	Stealing user credentials (email and password and/or access token)
Conditions	-
System elements	All components
System response	Data is protected
Metrics	Number of credentials stealed

Table 13: QA05 -Security (Broken Access Control And Session Management)

5.6.6 QA06 - Security (Missing Function Level Access Control)

Another vulnerability related to authorization that we must avoid is the Missing Function Level Access Control vulnerability [55]. Having a strong authorization mechanism that does no allow access to actions that do not are meant for certain user groups is one of top priorities. To avoid this we also must only show users the actions they can perform.

ID	06
Title	Security (Missing Function Level Access Control)
Stimulus source	Malicious user
Stimulus	Trying to access forbidden API endpoints
Conditions	User does not have privileges to perform an action
System elements	Server components
System response	Action is denied and logged
Metrics	Number of unprotected actions performed

Table 14: QA06 - Security (Missing Function Level Access Control)

5.6.7 QA07 - Ease Of Module Development

One of the key aspects that based the decision of developing a new platform is the fact that the old system one was not properly structured and modularized. This was affecting the development of new features. By designing a new system, we want to make it modular and make sure it is easy to add and remove modules.

ID	07
Title	Ease Of Module Development
Stimulus source	Development
Stimulus	Adding or removing a module on the system
Conditions	-
System elements	All server components and web application
System response	Other modules continue their operation normally
Metrics	Time taken to develop a new module. Time taken to remove a module from the system

Table 15: QA07 - Ease Of Module Development

5.7 Wireframes

With all the requirements defined, we designed some wireframes made with the purpose of visualizing the interface we wanted to obtain.

A wrireframe is a sketch of a screen or portion of it that intends to represent the layout and item placement of the final application interface. it will serve to guide us in the interface design process. This is an important step because the interface will be used by many people every day. We had to ensure that a proper and coherent interface was delivered.

Taking the most important screens that we will implement, we will define an adequate design for each of them. To note that the interfaces here presented are not final and can be changed at any time when user feedback justifies it.

5.7.1 Login screen

In Figure 20 it is shown the login screen for any user that is not authenticated. It is only a form where users can input their email and password. It is in fact the only page an unauthenticated user can view.

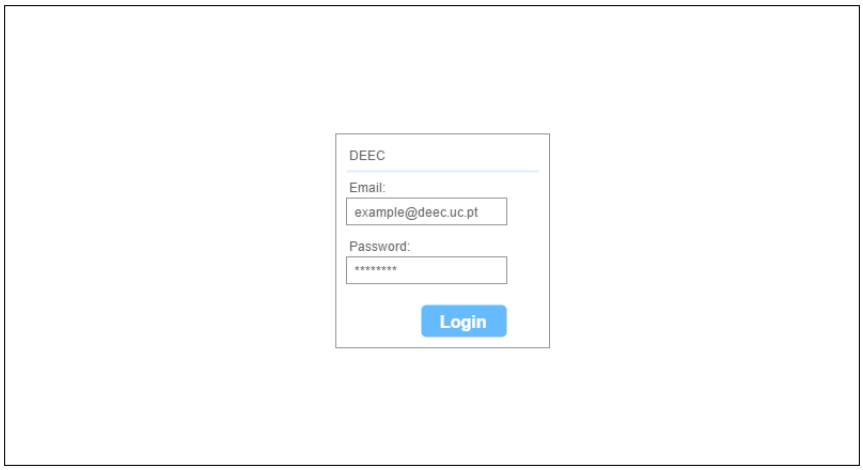


Figure 20: Login screen wireframe

5.7.2 Basic navigation menus

In Figure 21, we show the basic navigation menus. The side bar is where all URLs to the features of the platform are. The user name and logout button are also on the side bar. On the top bar the current page name is shown. There is also a button to access the notifications (if there are unread notifications the number of notifications is shown on the icon). Finally the language selector is on the upper left side.

The page content is where the current page is displayed. For simplicity, the wireframes after this one will not include these navigation menus.

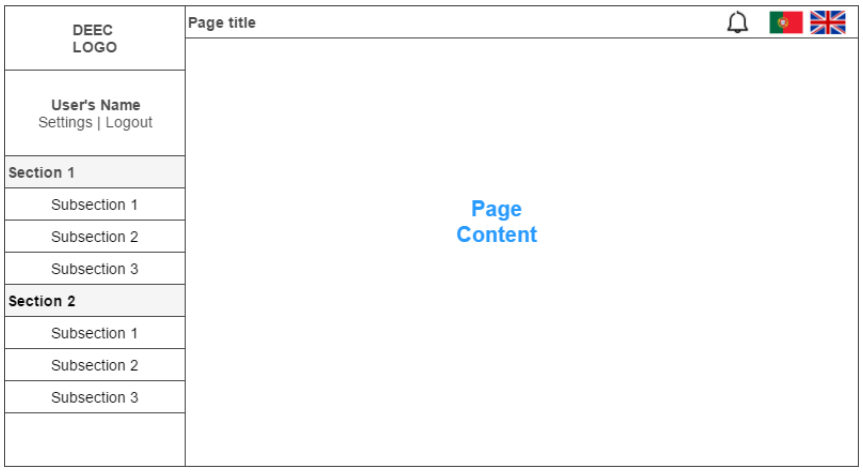


Figure 21: Basic navigation menus wireframe

5.7.3 Dissertation list screen

In Figure 22, we can clearly see the filters on the top of the screen. It is supposed that they are always visible. The number of the dissertations on the screen is supposed to vary according to the screen size of the user's device.

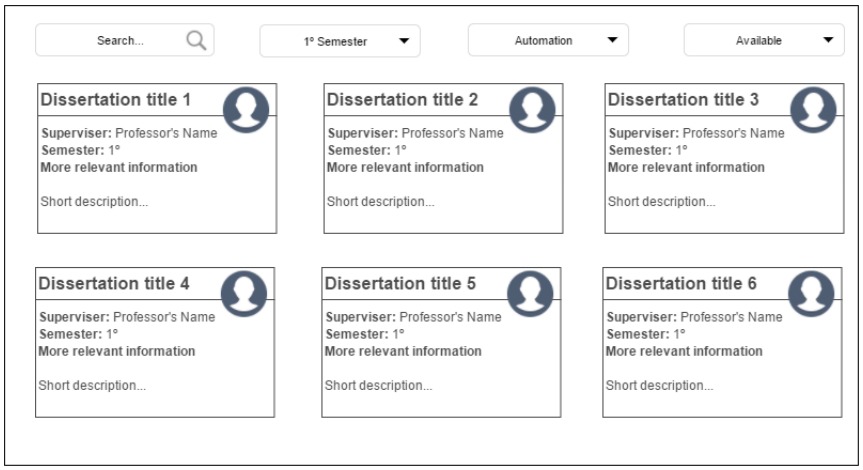


Figure 22: Dissertation list screen wireframe

5.7.4 Course creation screen

In Figure 23, we can see the form for the insertion of a new course, including all the fields with attributes of the pretended course.

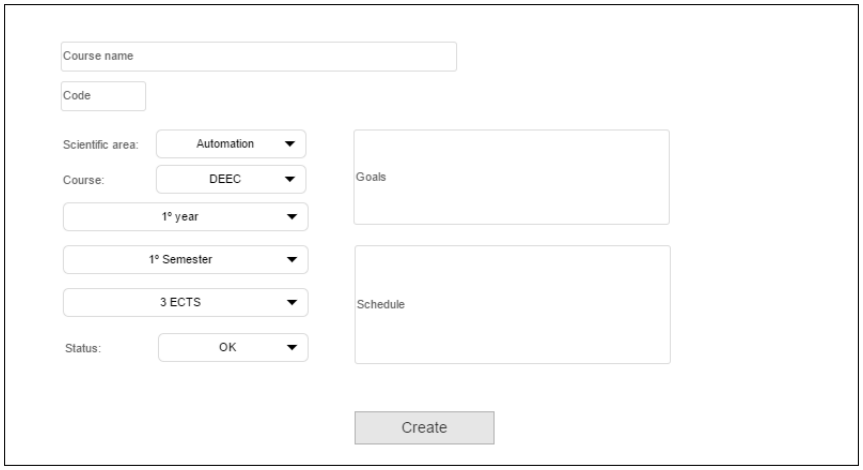


Figure 23: Course creation screen wireframe

5.7.5 Building view screen

In Figure 24 we can see the aspect of the building view. Here is where users can view information about the department building. There is the plant of the building and important points such as rooms and elevators are marked with green dots. When users hover a dot, details about that location are shown.

Important places can be searched and users can change the floor they are viewing.

Figure 24: Building view screen wireframe

5.7.6 Screen flow

Because the fact that the web application will have so many screens, in Figure 25 we present all the application screens and the flow between them. There are important aspects when viewing this diagram:

- The flow usually starts at the **Login screen** (if the user is not authenticated) or the **Dashboard screen** (in case the user is already authenticated)
- Screens colored with blue are accessible from any part of the application
- The **Logout** function is accessible from any part of the screen

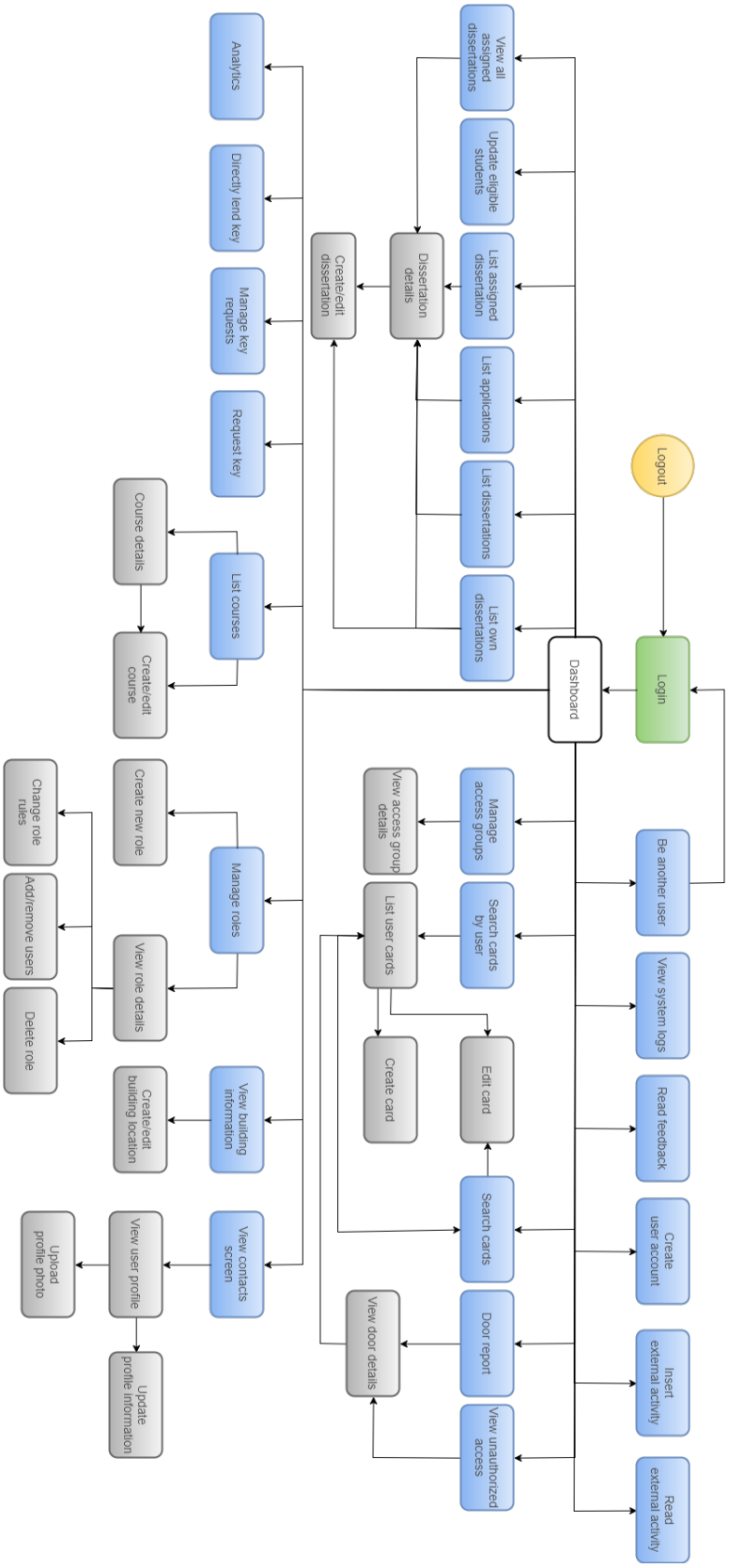


Figure 25: Flow between all the application screens

6 Architecture

A suitable architecture is a must for a solid system. We have defined what the new platform will do. Now we will define how we will achieve that.

In terms of architecture, we will first focus on the server components, next on the client application and finally on the database.

6.1 Architecture overview

We will first start by giving an overview of the architecture we chose. We were inspired by the C4 Model [56] in order to design our architecture representations, but it was not followed at its full extent.

As we can see in Figure 26, all users will interact with the platform using a web application, built with the AngularJS framework. This web application communicates with the server components (built with the LoopBack framework) to obtain and manipulate data. Our server components also communicate with other external services such as databases, LDAP, mail servers.

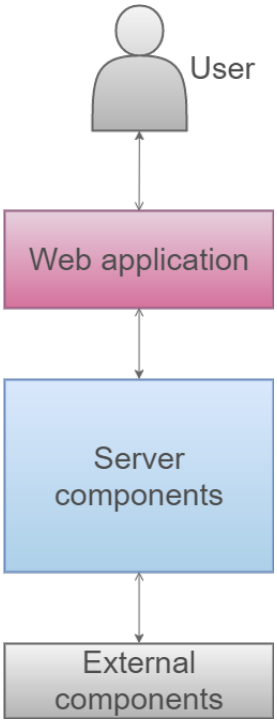


Figure 26: Architecture overview

Ensuring the security of all the user data is one relevant point when developing the architecture. One way to ensure that is to have all the communications between the web application and the server encrypted (HTTPS). Furthermore, communication with external services must also make use of secure protocols. A digital certificate was generated for our server **my.deec.uc.pt** and its purpose is to let users check the server authenticity of messages and allow us to use the HTTPS protocol.

6.2 Server architecture

Our server architecture is composed of the components we have to develop and have control (Server Components) and the components whose technologies already exist and we have no control (External Components).

To ensure our components can be accessed by our users we have the NGINX web server routing all traffic to our components. Our main component is the API and is this component that deals with all business logic. The Notifications Server component enables us to send web notifications and email notifications. The Content Server is a platform to store and retrieve files (upload photos by the users).

In theory all the Web Server Components should be deployed into different machines (like it is shown in the image), also including machine replication and load balancing using NGINX. But because of restrictions for this project we have no control over the physical layer and can only deploy them all in one single machine.

We use the already described tool PM2 to ensure that our components are always available to our users and to provide us with a better server monitoring. This tool also allows us to scale the instances of each component according to the user demand. At least 2 instances of the same component must run at the same time to maintain availability in case of failure of on of the instances.

All of the Server Components will run on the already created virtual machine *beta.deec.uc.pt*.

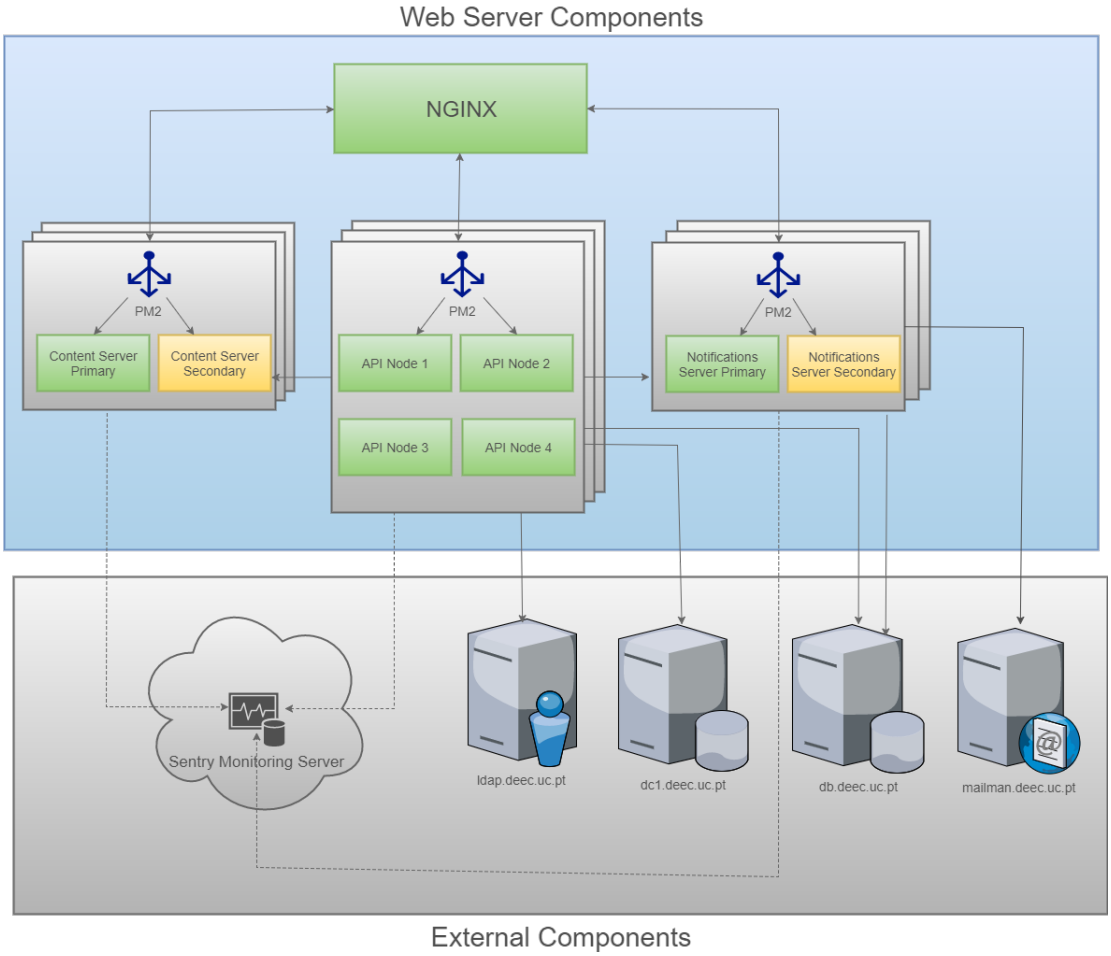


Figure 27: Proposed server architecture

In Figure 27 we can clearly see the different components our platform is composed of. There are two new external components that we will interact with:

- **Database *dc1.deec.uc.pt*** - This database stores all the information related with the new card access management software of the department. We use it to provide a better, better and intuitive way of dealing with the card management from within the new backoffice.
- **Sentry** [57] - Sentry is a service that aims to record problems of the software. When an error occurs in our production environment, its information is sent to the Sentry servers where we can analyze errors and manage them.

The other components were already explained and are intended to be maintained with the new backoffice.

We will now describe in detail the architecture of each component developed. Implementation details will be presented in the next chapter.

REST API

The REST API is the main component of the platform. This API is developed using the LoopBack framework. Some API endpoints were automatically generated by the framework and the remaining were developed by us to provide extra functionality. And

of course we had to perform several validations to the automatic generated endpoints to ensure that the API is properly secured.

Our API communicates with the database as shown in Figure 27 to map the tables to LoopBack models. It also communicates with the LDAP server to perform user authentication.

For authenticating a user on the API, the credentials received on the Login endpoint are sent to the LDAP server to verify if they match. Then, by using the LoopBack built authentication features, an access token is generated for that specific user.

For authorization management we used the LoopBack default authorization mechanism and extended it to allow dynamic assignment of roles and rules.

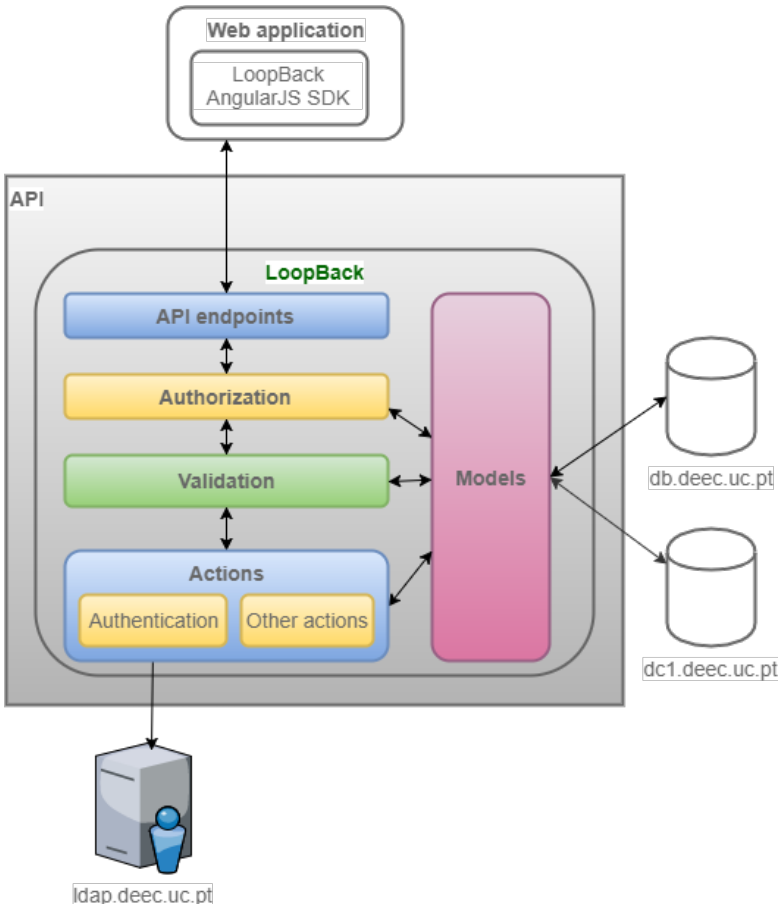


Figure 28: API architecture

In Figure 28 the details about the API are shown. When a request is made, it first reaches the endpoint specified, then checks the authorization, then the data is validated, and finally the desired action is performed on the model. The entry point for any unauthenticated user is the authentication method. It connects with the LDAP server to check the credentials of the user.

Notifications Server

This component is responsible for managing email and web notifications. When an important action takes place in the platform, the API send a notification request to the Notifications Server. Every notification is associated with at least a user and contain relevant data for the correct displaying of the notification. There are two types of notifications:

- **Email notifications** - Notifications directly sent to a user email. They are sent through the DEEC email server (email.deec.uc.pt).
- **Web notifications** - Notifications sent to a specific user browser through the Socket.io framework.

Web notifications are sent to users if they are online, otherwise they are stored to be sent later when the user becomes online. Notifications will always be stored in the database.

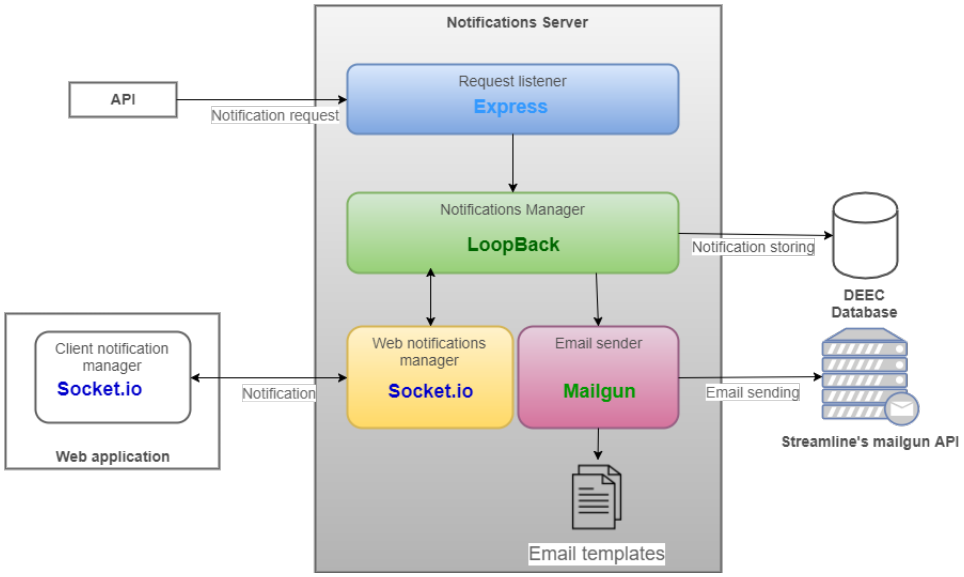


Figure 29: Notifications Server architecture

In Figure 29 we show the architecture of the Notifications Server. When a notification has to be sent, the request is received by the **request listener**. The listener is an HTTP endpoint and the request is a POST request to that endpoint.

Notifications are stored in the database using the LoopBack ORM.

To send emails we use **Mailgun** [58] which is a library that connects with the email server and helps with the protocol of sending an email. Templates were created to make email organization simpler.

Content Server

The Content Server is the component that enables the upload of files by the users and allows the download of those same files. When for example a user wants to upload a profile photo, that photo is uploaded to the REST API by the web application. Then,

the API gets the uploaded file and sends it to the Content Server. In the Content Server, the file is validated and converted if necessary. Finally it is stored on the file system and its meta data is saved to a NoSQL database.

In Figure 30 the architecture of the content server is defined. This solution is based on he prototype we have developed in the first half of this project.

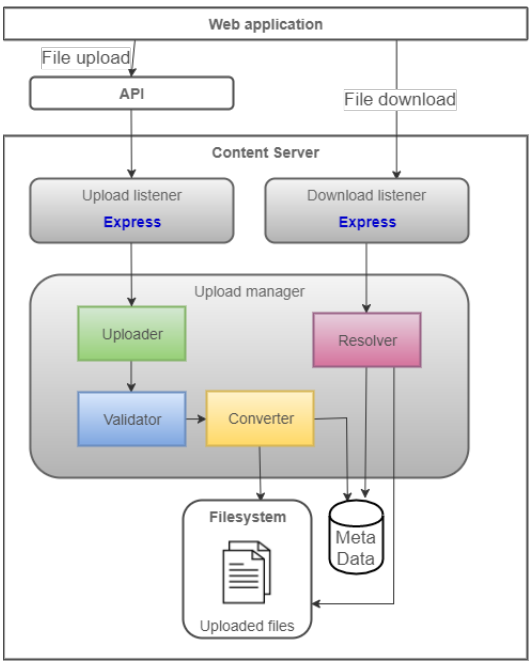


Figure 30: Content server architecture

6.3 Client architecture

The client is a web application that will run on any modern browser (including mobile browsers). Users interact with the application and are allowed to perform all the actions they are authorized.

We took advantage of the Angular Model-view-controller (MVC) architecture and the LoopBack SDK. The interface is developed using the Materialize framework. In Figure 31 we show the several types of components of the web application.

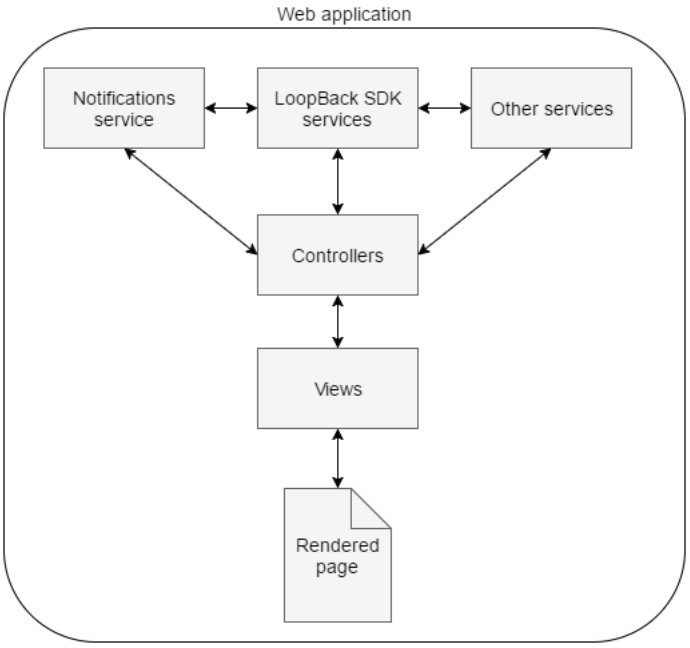


Figure 31: Proposed web application client architecture

LoopBack SDK Services

These are the services that are generated by the LoopBack framework. They are strongly used by our application to interact with all the API endpoints to get and modify data.

Notifications Service

This service is responsible for the communication with the notifications component on the server. When the user successfully authenticates on the web application, the notification service will also authenticate with the server. After that, it listens for incoming notifications and passes them to the other components of the application.

Other services

Other services were developed to provide two types of functions:

- Data storing: temporary data store of the results obtained by the API. This ensures that the API is not overloaded with requests. Not all results can be stored, because they constantly change and must be always updated.
- Global methods: methods that are used across all the application are provided by our custom services.

Controllers

Controllers are responsible for interacting with the services and also interact with the application views and control their data. Our objective is that a controller is assigned to only one view and control only its data.

Views

Views are where our interface is defined. Each screen we showed earlier is a view in this architecture. Views get the variables from the controllers and draw the interface using them. Views usually contain HTML tags and AngularJS directives.

Rendered pages

The rendered pages are actually the result of the rendering process of the AngularJS framework. It is the final result of a view and the page that the user sees. Because they are the result of the views, rendered pages must reflect the wireframes we have shown before.

6.4 Modularization

One of the purposes in developing a new system was to make it modular. We had to make sure that when developing the client and server, the code would be well organized in modules.

A module has a set of features that share a common context. There are modules that are the base of the application and modules that add features to the application.

Modules include client and server code. So when a module is deployed, it has to be done in the client and server. This is a must to ensure that the module is well deployed. In Figure 32 we present the modules of the platform. Blue modules are the base of the system.

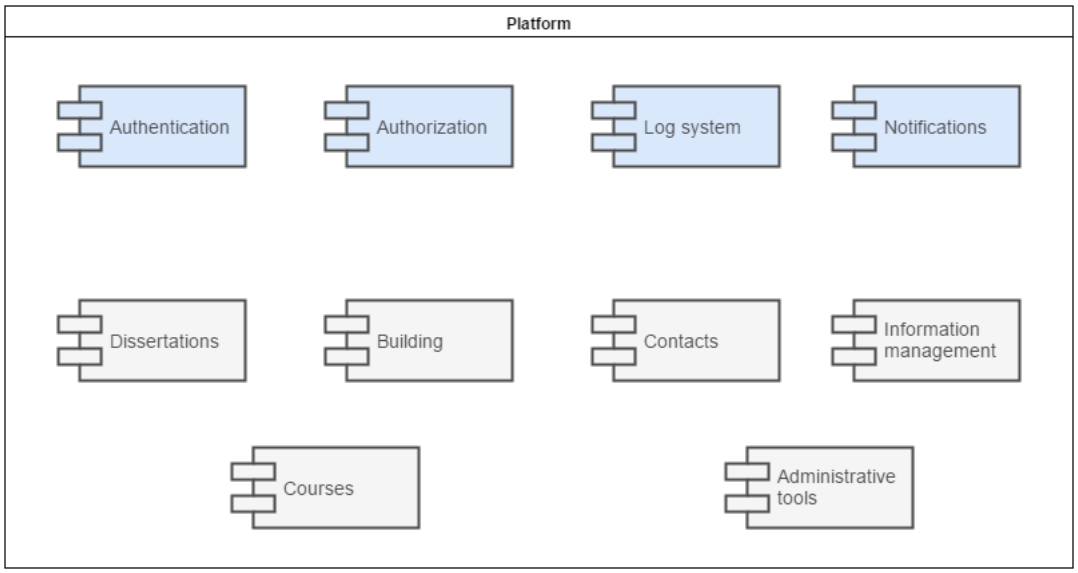


Figure 32: Modules of the system

6.5 Database

To implement the new requirements we have to make some changes to the database. The old backoffice database serves its purpose, but as we saw before, it has several tables that

are not in use. So, it does not make sense to continue with those tables in the database. We will remove them.

Because we have two distinct user tables (*aluno* and *pessoas*) we feel that it is necessary to create a table named **user**. This table is the base user table that has the *aluno* and *pessoas* as children. This way, when there are features that are used by both *aluno* and *pessoas* (e.g. authorization, access cards), we can reference them by their user_id.

Primary keys were added to each table and relations were also established.

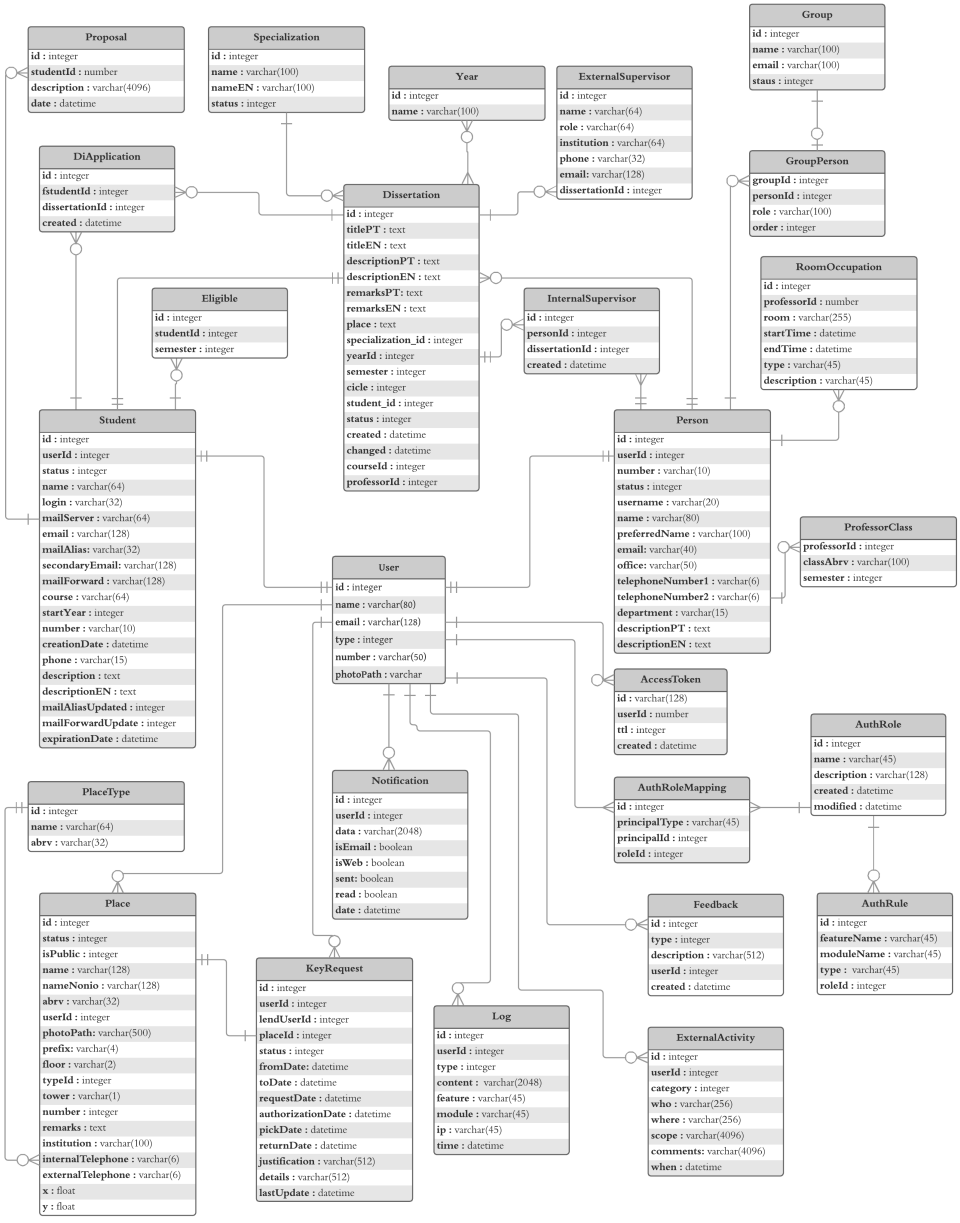


Figure 33: New tables to be added to the database

In Figure 33 we show the ER diagram of the proposed database structure. Those are the tables required to implement the proposed features. The tables that already exist are omitted except the table *aluno* and *pessoas*. There are some tables missing: those that are related to the card management software we described before. We will not show them here because although we will be using them, they do not actually belong to us.

When the old Backoffice is no longer active, all table names and their columns can be normalized to be in the same language and follow the same patterns. The way LoopBack maps tables into JavaScript objects will ensure that if a table or its columns change their name, only the model configuration file has to be changed in order to reflect those changes.

7 Implementation

In this section we will describe in detail the steps taken to implement all the components in our platform. These steps include the decisions taken and the problems we had to solve in order to build the software proposed in the last chapters.

From all the development performed, we chose some of the most interesting and challenging decisions/problems we had and how we solved. We will begin by presenting the process of the database migration, then describe how we developed the different components, performance optimizations made and finishing of by showcasing the final application.

7.1 Database

The database we used is the one being used by the old backoffice. It is a MySQL 5 Database and stores all the data we need, but of course we had to make some changes. We started by migrating the data we wanted and creating new tables according to the ER in Figure 33. We then checked for problems in the database and solved them.

7.1.1 Migrating data

As said before, we had to use the database already being used by the old backoffice. We already defined what are going to be the new tables and what changes we have to make. So, we created a migration script to run the first time the API server starts, and checks if any migration is needed in order for the system to run properly. The script follows the next steps:

- **Check the need of migrations** - Verify if the users table exists and has any data (if does no exists, means it will have to migrate data)
- **Create new tables** - The new tables defined in the ER presented on the last chapter are created, using SQL scripts.
- **Map persons and students to the users table** - Data (name, email) from tables **pessoas** and **aluno** is merged in the **User** table and saves the user_id in each of those tables(person, student). When developing the script, we had to manually check every person account to assign them a role. Hence, when the migrator runs, it will automatically assign them the role. For the students, we assigned them all the **student role**.
- **Migrate supervisors** - Since the table containing the dissertations data now contains the supervisor_id, the script gets the supervisor_id from the supervisors table and fills it in the dissertations table.

We also created several triggers in the database. When a person or student account is created, it automatically creates a user in the user table, and vice-versa.

We had to test this a few times to make sure it would not break the production database. We tested it using cloned data from the production database. We even tested that if it

stoped at the middle of a migration he would then continue where it had stopped causing no consistency problems.

7.1.2 Database encoding

When we started to analyze the database and the data it contained to develop the migration script, we noticed that the database and the tables had the Latin1 (ISO8859-1 [59]) encoding. This was oddly strange, because when displaying that same data in a web page it was correctly shown, even though the page encoding was Unicode (UTF-8 [60]). To try and solve this problem, we followed a method [61] to discover if we needed to convert our entire database data from Latin1 to UTF-8. We discovered that in fact the data was already stored in UTF-8 format. We only changed the encoding in all the tables and database using their respective settings. The explanation for this "problem" is that even though the table settings were telling us the data should be stored in Latin1, the old backoffice was storing new information in UTF-8 and the database did not force the encoding, thus leading to encoding inconsistencies.

7.1.3 Integration with the card management database

Having to use a third party database by another software was necessary to implement the *card management* feature. The database uses Microsoft SQL Server [62] and had to be integrated in our API, by importing the tables we needed into LoopBack models. This process took some time due to the analysis of the *card management software* and its database. We had to make some tests in order to be sure which tables we had to use (the database has almost 200 tables). For example when adding a new card or changing its rules in the *card management software*, we would verify which tables it would change. The same for adding new users to the software.

We had to get a way to relation our user model in our API to the card access user model using a foreign key. To do this we used the user number (person identification number assigned by University of Coimbra). We could not use the backoffice user id in this case, because there were already users created in the card access database and it would cause inconsistencies.

7.2 API Server

As said before, for the API Server, we used the LoopBack Framework. It makes the API development process easier due to its automatic generation of models, default API endpoints (GET, POST, PATCH, DELETE) and its ORM. When we developed the prototype in the first half of the project, we noticed that due to the large amount of models and methods we had to implement, it would not scale well in terms of code structure and maintainability. We researched for solutions for this problem, but did not find reliable methods to solve it. We had to implement a set of solutions to make the development process easier and to help the future developers. Those solutions are presented in the following sections.

7.2.1 Making the LoopBack Framework modular

The Loopback framework we are using provides us with a great set of features. Nevertheless, it lacks the concept of modules. Upon researching for solutions for this problem, we could not find a one for this particular problem, we found some extensions to the framework, but they lacked documentation. For that reason, and to make our application modular, it was necessary to implement the concept of modules on top of the stack we are using. The fact that we can divide the application into modules and features helps the development process and as we will see ahead, it also makes it easier to handle the dynamic authorization that is an important part of this platform.

The first step taken was the analysis of the framework and identification of the points were the application logic is usually implemented. We have that for each model (User, AccessToken, Dissertation, etc) there is its associated logic. For smaller platforms it fits the purpose well. However, since we have some complex logic involving more than one model at the same time, we think that is better suited and makes more sense to have modules involving various models. In addition, a model has several types of Access Control Lists. As an example, we can specify that a particular type of user can create a new instance of a model.

The proposed approach to this issue was to create a module loader. This module loader is responsible for, at the very beginning of the boot process of the API server, loading all the detected modules.

Every model is a directory that contains a configuration file and the its features. The configuration file contains the module name, a flag stating if the module is public our private and all the features the module has. A public module means that its features can be accessible though the API, private means that the features can only be accessible by other models within the server.

Each feature is represented by a name and all the ACLs that it needs to operate. This is where this approach makes sense, because we are stating that a given method can interfere with various models and perform different actions on them. An example of a configuration file is shown bellow:

```
{
  "name": "cardAccess",
  "isNavbar": true,
  "isPublic": true,
  "description": "Module responsible for the card access
    ↪ management.",
  "features": [
    {
      "name": "viewUnauthorizedAccess",
      "run": false,
      "isPublic": true,
      "isNavbar": true,
      "defaultRoles": [
      ],
      "acls": [
```

```
        {
          "model": "accessAlarm",
          "accessType": "READ",
          "permission": "ALLOW"
        }
      ]
    }
  ],
  "dependencies": ["apiUtils"]
}
```

When the API boots, the module loader will check the **modules** directory for all the modules. For each of them it checks if it has dependencies and then loads them according to their priority (assures that a module dependency is load before the dependant module). When loading a module it will create the ACLs defined and perform some init actions (if applicable). All the features are saved in a in-memory object to easily be retrieved at any time by the API.

7.3 Handling authentication

Authorization on the platform is performed by checking the user credentials on an LDAP server that contains all DEEC accounts. Default LoopBack authentication is performed over a database that contains all user account details (emails, usernames, passwords). At first we tried to use a connector named **passport-ldapauth** [63] to perform the authentication, but it required a large amount of configuration and we could not manage to make it work. So we decided to override the default LoopBack authentication method. The default authentication method is the **login** function on the **User** model. The login method we implemented is the following:

1. Check if the credentials are valid - Check if the email and password are not empty.
2. Search the database for an account with that email - If an account is not found, that means the credentials are invalid and the process return with a status code of 401.
3. Get the account username. The username is the first half of the email (the local-part).
4. Perform a bind to the LDAP server (using the library **ldapjs** [64]) that bind uses the username retrieved before and the password the user sent.
5. If that bind returns a result, that means the password is correct and we can authenticate the user.
6. An access token (64 byte length) is created using the default LoopBack method. That access token is created with an expiration time of 2 weeks.
7. The access token is sent to the user and is used in future request to identify the user.

An user can have multiple access tokens. That means they can have sessions open in various devices at same time. Sessions can be closed at any time by logging out on the device they want. When performing a log out, the access token is deleted which means that session terminated.

7.3.1 Handling dynamic authorization

The main goal of having dynamic authorization is that we can have a better control in who can access the features. This dynamic authorization was thought to be a simple way of letting the system administrators to manage the authorization of the application without the need to change the source code of the platform. Simply put: users belong to one or more roles and roles have as many rules as required. One rule dictates if the associated role has access to a certain feature.

We made a user interface within the web application that can be accessed by System Administrators to modify rules regarding roles, users and rules. When any change is made, it the API receiving the modified rules, signals other API nodes to reload the propagated to all API nodes, ensuring that consistency is maintained. In Figure 34 we can see the page where system administrators can modify the features a certain role can perform.

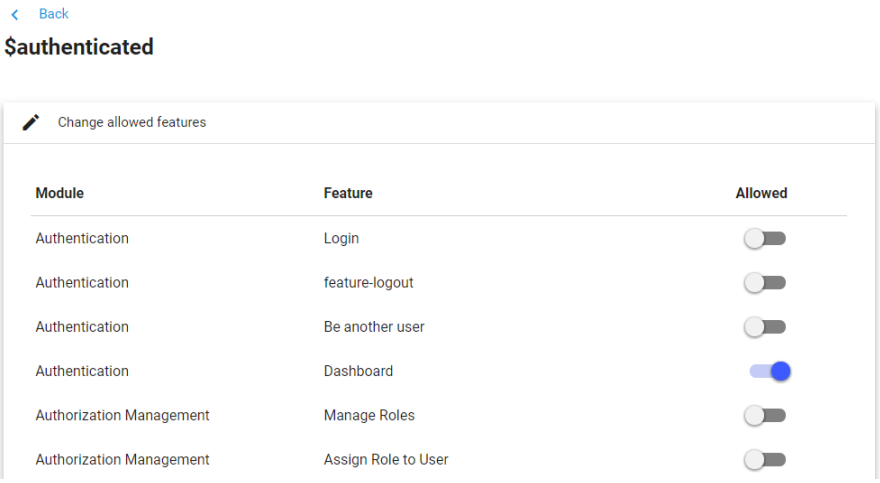


Figure 34: User interface for the dynamic authorization feature.

By assigning roles to features through rules we also solve the problem of users knowing what can they perform. For example: when the web applications opens, what features will it show for that particular user? The solution was to have an endpoint (*/api/allowed*) in the API that when requested, will reply with the features available for that user. This endpoint is always requested at launch of the web application, ensuring that the users always see the actions they can perform. If the user is not authenticated the endpoint replies with features available for the $unathenticated role (default LoopBack role). We discuss the web application authorization methods in detail in section 7.4.2 of this chapter.

7.4 Web application

As said before the web application was developed using the AngularJS framework for the front end logic and Materialize for the front end interface and visual. After all development we have developed about 47 controllers, 16 services and 24 directives. The number of controllers essentially matches our number of functional requirements, this is because we assumed that one controller is responsible for one requirement and directly matches its flow. These controllers communicate with the services that old the temporary data in the browser and are responsible for communicating with the server to manipulate it. The amount of services also reflects the amount of modules we have. Of course we divided some modules in two, helping us have a more organized source code.

We now explain some of the challenges when developing the web application.

7.4.1 AngularJS and ES6

When we started developing the prototype web application we were a bit disappointed by having to use an old JavaScript version. This was because browsers typically do not support the recent versions as soon as they are released. For example, classes are not yet supported by Internet Explorer 9 [65]. We felt that implementing a full web application without recent features would make our code much less readable and structured.

Thus, after researching, we decided to use two tools that we think helped us a lot to organize the code: Babel [66] and Webpack [67].

- **Babel** - Babel is a tool that transpiles code into another version of the language. For example, translating ES6 JavaScript into older versions of the language, supported across all the browsers.
- **Webpack** - Webpack bundles a set of source files into a single one. It can also apply minification/obfuscation to the code. Used together with Babel, it can transform all our JavaScript/CSS/HTML files into a single bundle that will be supported by all browsers. This bundle is tipically a JavaScript file that is loaded when the application starts and then unpacks all the necessary files and runs them in the browser. We also used the minification plugin included in the Webpack workflow, this allowed us to decrease the bundle size from about 6 Megabytes to about 1Mb. This will be addressed in the optimization section (7.7.1).

7.4.2 Dealing with authorization

We explained how we handle the authorization on the API. But we also had to address it on the web application. As we saw, there is an endpoint were the allowed features for an user can be retrieved. So, we made a mechanism that when the web application starts, the first thing it does is to retrieve those allowed features from the server. Then, when a user clicks and URL to change to another page, it will check if that page belongs to an allowed feature. If it is not authorized, it will redirect the user to the dashboard page, or to the login page if the user is not authenticated.

We also made sure that the navigation bar only displays features the user can perform. We implemented a way to configure the navigation bar using a configuration file written in YAML [68]. This configuration file defines the menus and sub menus, their order, icons, colors, what feature they represent, and what is the state they go when clicked. Below we can see a simple example:

```
- name: Dashboard
  icon: dashboard
  standalone: true
  state: app.dashboard

- name: Contacts
  icon: people
  state: app.information-viewContacts
  standalone: true

- name: Dissertations
  icon: school
  subsections:

    - name: List Dissertations
      module: dissertations
      feature: listDissertations
      state: app.dissertations-listDissertations
      icon: list
```

When the web application gets the allowed features from the server, it then generates the menu bar based on the configuration file defined.

7.4.3 Multiple language support

Since our application must support at least the Portuguese and English language, we have made a simple Angular filter to translate the required parts of the application depending on the language the user uses. Those language preferences are saved only on the client side and persist in the local storage of the browser.

Translations are stored as YAML objects that are then converted to JavaScript objects. Every translation has a token and its corresponding translation. Tokens are then used across the application views.

At some point when developing the web application we noticed that it was hard to keep track of the tokens that need to be translated. This is because we did not translate while we developed. As such, lots of tokens where being left without translation. We know that this was a problem that could be avoided if from the begging we kept track of the translation tokens. To solve this problem, we have built a simple tool that helps translators find untranslated tokens and prompts for the correct translation. The way it works is by traveling all our web application source files and using regular expressions (JavaScript RegExp [69]) to discover translation tokens. It then compares the tokens

discovered with the already translated to verify if a translation must be created. If a translation is created, it is automatically stored in the translations file.

7.4.4 Location selector and browser

Information about the building locations already exists in the DEEC database but we wanted to make it more visual to users. The idea presented in the requirements chapter was that locations would be placed in a map for easy visualization. We decided to add Cartesian coordinates to the Location model. The point **(x,y)** indicates the point the location is on a map from the top left corner. Both values range from 0 to 1. We chose to normalize the coordinates to allow us to render the map in different screen sizes.

To exemplify this explanation, we can see in Figure 35, three cases where we map a point into a map. The red dot indicates the point and has its real and normalized coordinates. When the user clicks the map to select a location, it will get the real coordinates using the JavaScript mouse event coordinates relative to the map image. Those coordinates are then normalized using the formulas ***x = real_x / image_width*** and ***y = real_y / image_height***. These normalized coordinates are the ones being saved in the database. The reverse process is when a user wants to view the locations. The normalized coordinates are converted to real coordinates following the formulas ***real_x = x * image_width*** and ***real_y = y * image_height***. This method ensures that a point is always in the same place, independently of the image scaling or ratio as Figure 35 shows.

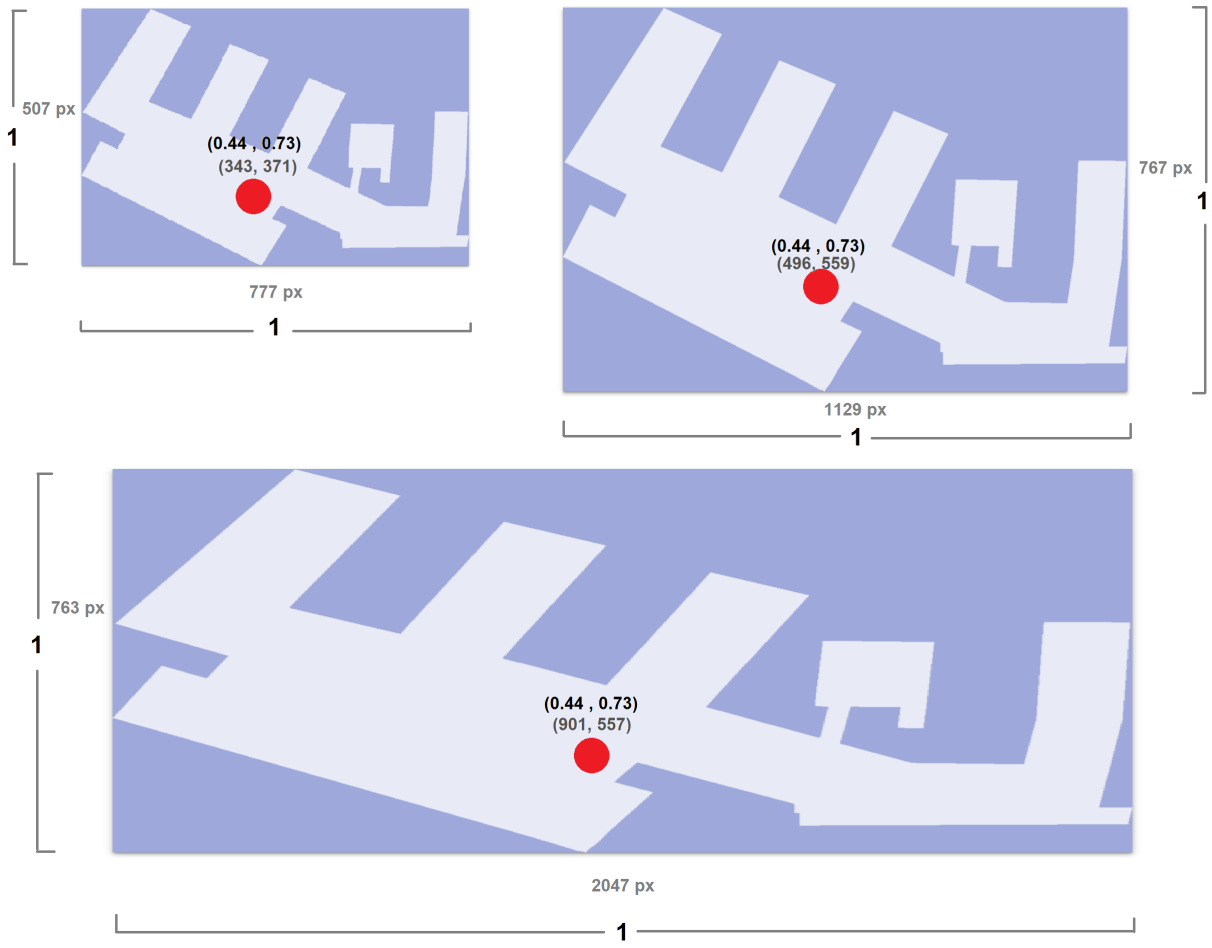


Figure 35: Coordinates normalization visualization.

7.4.5 Monitor web application usage and statistics

This next part does not correspond to any functional requirement but is big helper to monitor the web application usage. We know the log system can help to monitoring too, and it helps, but it only displays actions that happen on the server side (data changes, errors, important actions). We want to have a better understand at the users behaviour (the pages they visit the most, time spent at the pages, number of users, etc).

For this particular feature we used AngularJS interceptors to detect when a user goes from on page to another. We store that page change in our servers as an event, containing relevant information (page name, timestamp and a UUID [70]). The information is sent to the server using the web socket communication between the browser and the notifications server. We used this connection because it is always available, and this way we do not saturate the API server with requests. Although this method of getting user behaviour on the web application works when using the AngularJS, it cannot be used by other frameworks. If in the future the developers switch to another framework or develop a mobile application for MyDEEC, they must make an interceptor for that new setup. Despite that, the server logic will remain the same.

Only system administrators have access to this analytics feature, but because our architecture supports the dynamic role management we could have created a role called **analytics supervisor** that could use this feature. Figure 36 shows the analytics feature where statistics are presented. In the next chapter we will analyze the obtained results for the actual platform.

These statistics are retrieved from the server. They are generated when requested and follow a simple statistical algorithm that analyzes the events occurred between two dates specified. The results are then returned and displayed using charts. There are 5 types of metrics the data is represented:

- Views per page - This is simply the amount of events that transition to that page.
- Average time per page (in seconds) - To calculate this metric, the algorithm first has to calculate individual times for each page. For example: if a user goes from page A to B and from B to C, then there are three events (entering page A, entering page B and entering page C). Each event has its time of occurrence, so we calculate the time at page A by subtracting the time of the transition to page B, minus the transition for page A. Adding all the times for that page A from the various users we get the total time for the page, then we divide it by the total number of views for that page.
- Views per hour of the day (in percentage) - Calculated by verifying the hour of the day at which the events occurred and dividing by the total event count.
- Views per day of the week (in percentage) - Calculated by verifying the day of the week at which the events occurred and dividing by the total event count.
- Views per day of the month (in percentage) - Calculated by verifying the day of the month at which the events occurred and dividing by the total event count.

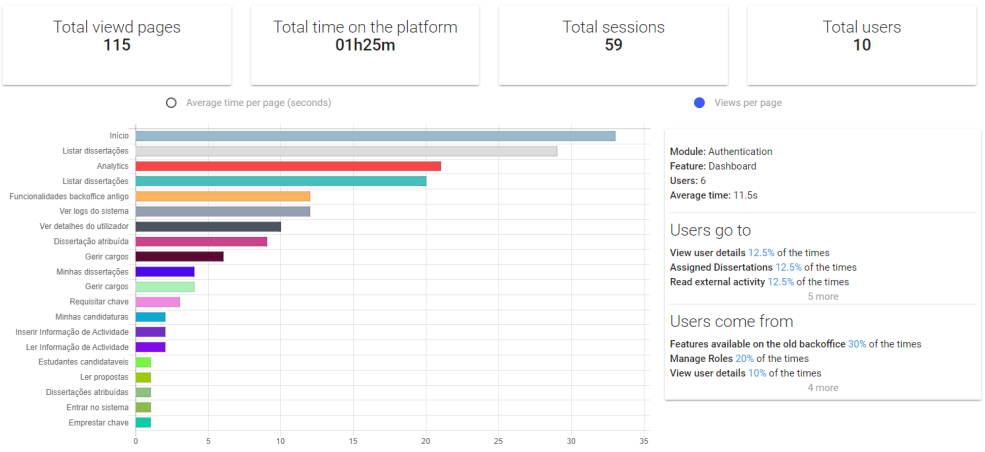


Figure 36: Analytics page.

The algorithm also uses the data to calculate paths users ussually follow to navigate though the web application. For example, what page is more probable to be visited next

if a user is in the dashboard page.

7.5 Notifications server

Developing a notification system that would be easy to send notifications to users was one of the requirements to be implemented.

The Notifications Server developed provides a simple API where it expects requests to send notifications. Authentication is made by using a pre shared access token. This ensures that only our API nodes can communicate with the Notifications Server and send messages. The endpoint */send* receives a notification request object that contains:

- **Notification type** - This must be one of the notification types configured in the Notifications Server (e.g. key requested notification, dissertation assigned requet)
- **User ids** - An array containing the ids of the destination users.
- **Is email notification** - If the notification is to be sent by email.
- **Is web notification** - If the notification is to be sent by web. To note that notifications can be sent by both email and web at the same time.
- **Data object** - Object containing the data that the notification needs to be rendered.

When a user launches the web application, it will try to connect to the Notifications Server and perform authentication. The communication is performed in the following way:

1. Web application tries to communicate with the Notifications Server using web sockets.
2. Web application sends user authentication to be checked.
3. The Notifications Server checks the authorization using the access token and searching it on the database.
4. Web application asks for the last notifications.
5. The server gets the notifications for that particular user.

A global object is used in the server to save the users online. This object is used when a notification has to be sent to verify if the destination user is online. If the user is in the global object, notification is directly sent, otherwise it is saved in the database for sending as soon the user comes online. Even if the user is online, the notification is saved on the database. When the web application closes and the connection is finished, the user is removed from the global object.

7.6 Content server

To enable platform users to upload files and download them, we created a simple content server. We use Express to serve the files over HTTP/HTTPS and a simple NoSQL

database to store the metadata about the files. We also use ImageMagick to convert image files.

Users do not upload files directly to this content server, they upload the files to the API Server and then it uploads to this server. Our API Server is the only allowed to change any data on this server. Authentication is made using a pre shared access token. That access token is sent in requests that change files.

Using the Express framework, we developed a simple API that enables the file management:

Each file has a static name and optionally a dynamic name. The static name is the real file name on the file system. The dynamic name is the name that the uploader assigns to that file when uploading. The dynamic name also contains the name of the folder that the file is in.

- **/upload/** (required authentication) - This method allows the upload of a new file to an existing folder. We will discuss the upload process ahead.
- **/createFolder/** (requires authentication) - This method creates a new folder that can be used in future request to upload files.
- **/delete/** (requires authentication) - This method deletes a file from the server.
- **/d/*** - Downloads a file by its dynamic name.
- **/s/*** - Downloads a file by its static name.

7.7 Overall performance

From the beginning of this project we wanted to make sure we had an optimized platform. We now show some of the optimizations we made to our platform.

7.7.1 Optimizing build file sizes

As we said before, we are using Webpack to generate our application's build file. It is this bundle of a single JavaScript file that is being transferred to the client browser.

The first builds we made were between 5Mb and 6Mb. Downloading a file with such size would be no problem in fast connections, but for clients with slow connections (less than 1 Mbps) or even for those who have limited data plans, it would make a huge difference. That was the reason we started to minify or entire source code using a Webpack plugin. Of course we had to change a few things in our source code (make sure libraries would still be available, injection of services in AngularJS was made in the right way). After this minification, the file's size was about 1Mb. This was a considerable drop in size. But we tried even harder to drop that size.

We started searching for files where we could reduce some code. The first place to look was obvious: the code we made. Although, it had no repeated code and even if we could optimize some parts, it would not make a big impact in the total bundle size. We then proceeded to the libraries (AngularJS, Materialize, and other smaller ones). We could

remove things that were not being used, but this would require to much knowledge of the source code of those libraries. And if in the future we updated one of those libraries, we would have to remove again unnecessary code.

At this point we started a search for tools that could help us discover any file that was making our bundle to have such file size. We found an extremely helpful tool [71] that allowed us to analyze our bundle and view the sizes of the several source files of our bundle. In Figure 37 we can see the composition of our bundle (without minification). Inside the green square is all the code we have developed. Inside the red square are all the libraries used by our code. And inside the blue square is the code generated by the LoopBack framework to use with AngularJS. We knew it was a big file, but because our application had grown so much in terms of features and modules, that file also grown to be about 30% of our entire application code.

We started to analyze the file **lb-services.js** and discovered several patterns. There was a lot of repeated code with only slightly changes (there was all the information relative to the API endpoints). We came up with a solution that generates the original *lb-services.js* file from a simple set of configurations when the application starts.

We also made a generator for that configuration file. The generator travels each model, and saves its endpoints configurations to an output file. It is that output file that is used at the beginning of the web application to generate the Angular services used to access out API.

This solution allowed us to reduce the size of the API services from 400kb to about 18kb by generating the services at run time. It has minimum impact in the web application performance but has a significant impact in the file size.

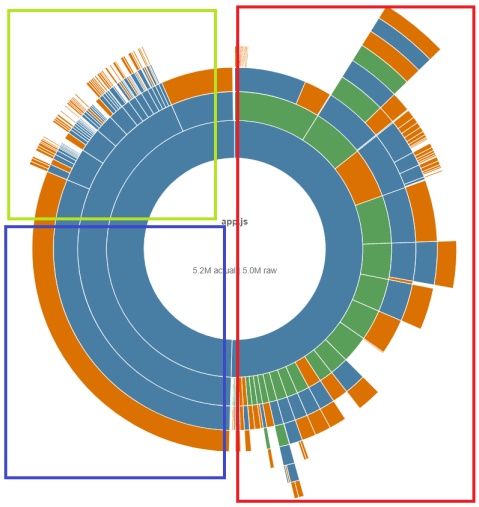


Figure 37: Web application size details

7.7.2 Web server configurations

To optimize the delivery of content to the users, we had to make some configurations in our web server. The following configurations were made:

- **Activating compression** - We activated gzip compression in the delivery of static files. This made our web application JavaScript bundle decrease its size from 1 Megabyte to about 400 Kilobytes.
- **Activating cache** - We activated cache in all the static files (HTML files, JavaScript bundle, images, fonts, icons). To the HTML files (particularly the *index.html*) we defined a expiration of about 1 day, this ensures that if we release a new version of the application, the browsers download it.
- **Bundle size versioning** - Because we are caching our bundle, when we updated the bundle file, it would not be served to the user browser unless that file was expiered in the user's browser. This is not a big issue, but to ensure our users always have the last web application version available, we assigned a file version to each bundle that is automatically assigned when the application is built. When we reference the file in the code we asign it a version. For example: *bundle.js?v=3*. The browser interprets this as a file change and forces to download it again, thus downloading the updated file.

7.8 Deployment

We used nodemon in our development environment, as it really improved the development process by providing hot reloading of the components. Although, we had to configure a deployment mechanism that allowed us to run the components in production mode to use production databases and other servers like LDAP and mail server.

To start a component in development, production or test mode, we pass them an environment variable that states the environment we are running the component in. Since we use PM2 to manage the components on the server, we made configurations that can easily be launched through a script.

For each component we created a script that does the following:

1. Runs *npm install* to install the dependencies needed for that component
2. Sets the NODE_ENV environmental variable to **production**
3. Calls PM2 to run the component in cluster mode

PM2 is configured to automatically restart a cluster node in case of failure. It is also configured to wait for a node startup in order to actually consider it running. This allows us to reload a component at any time (for redeploying reasons) without having any downtime. When reloading the component, PM2 shuts down one cluster node at a time to ensure that at least one node is available, thus preventing downtime (gracefully reload).

We can use the PM2 logging system to check for component errors at any time by running *pm2 logs <component_name*. This is not related with the logging system we made, that is a different feature intended to log the actions performed by users in the platform.

7.9 Web application showcase

Application screens were developed accordingly with the wireframes designed in chapter 5. We made extensive use of the Materialize CSS framework to maintain the same visual interface and behaviour. We will now show examples of the interfaces developed for our web application.

We now present some final screens of the web application.

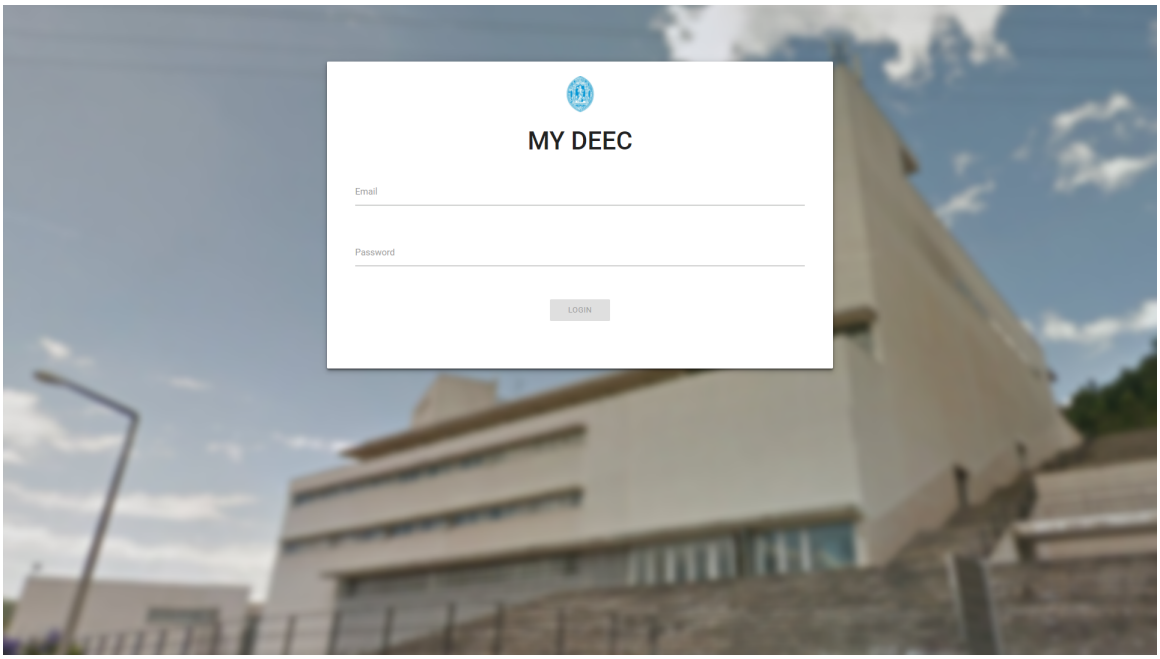


Figure 38: Login final screen.

Figure 38 shows the login page that every user sees when is not authenticated.

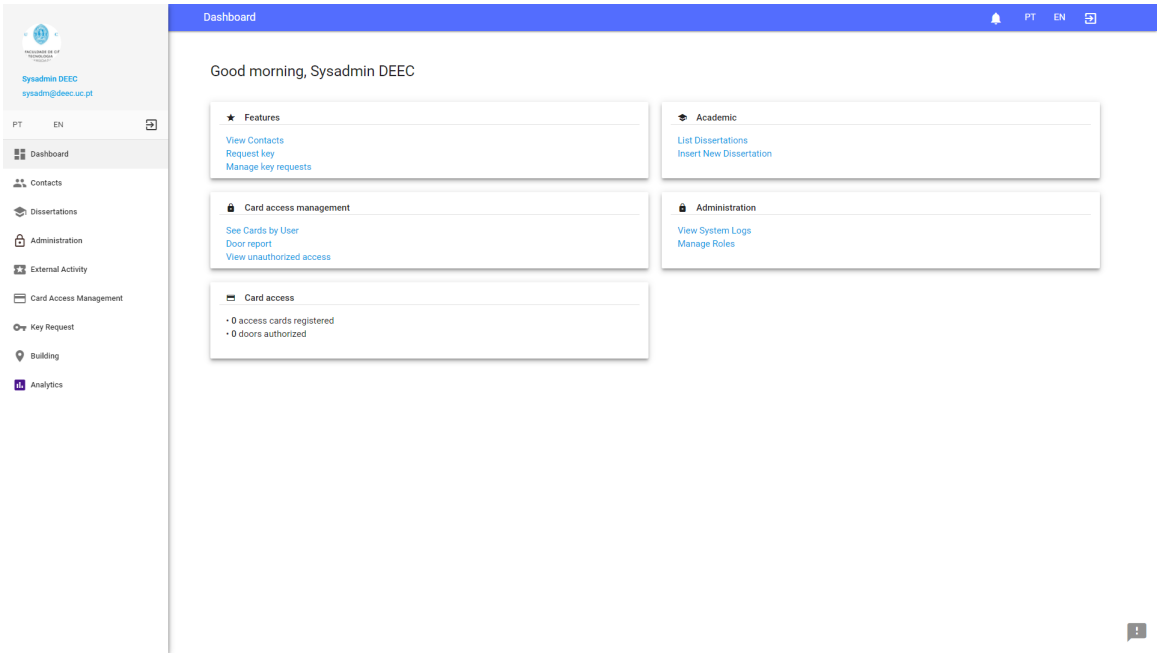


Figure 39: Dashboard final screen.

Figure 39 shows the dashboard that users see when they log in on the platform.

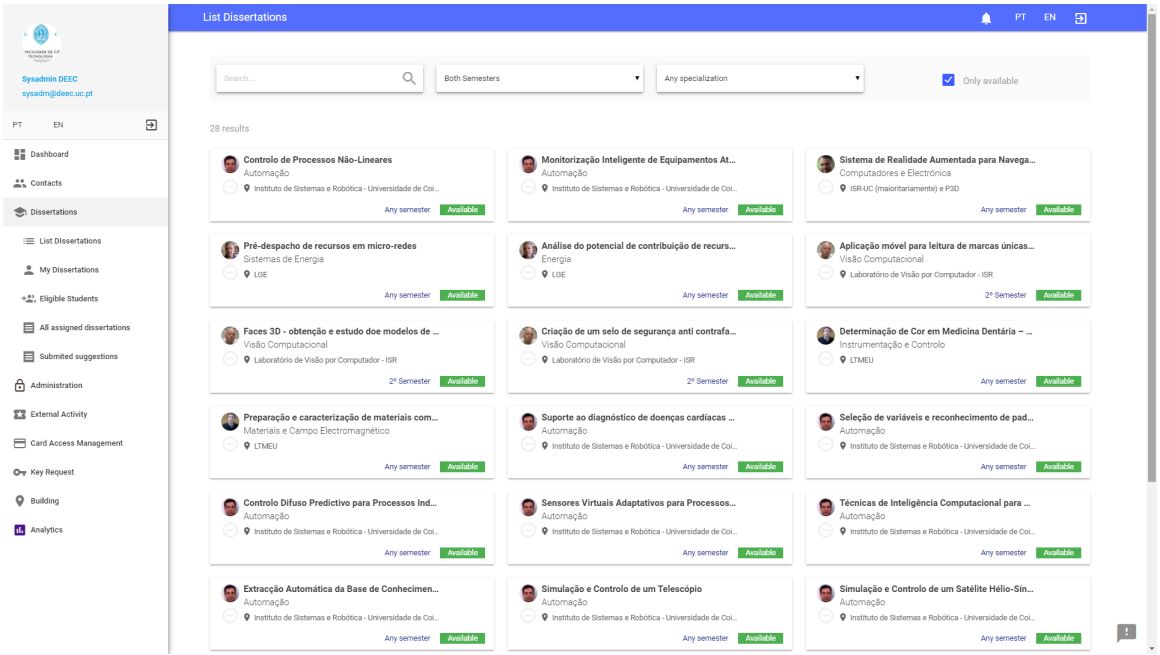


Figure 40: Dissertations final screen.

Figure 40 shows the list of dissertations that any user can see.

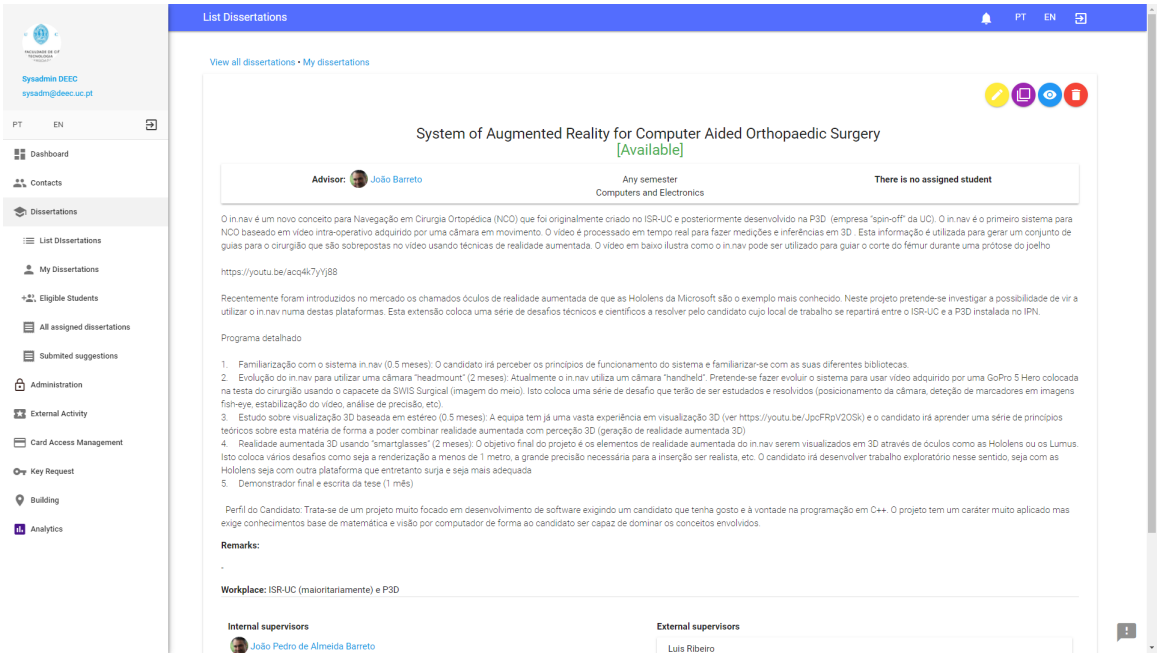


Figure 41: Dissertation details final screen.

Figure 41 shows the dissertation details screen that users see when they click a dissertation from the list.

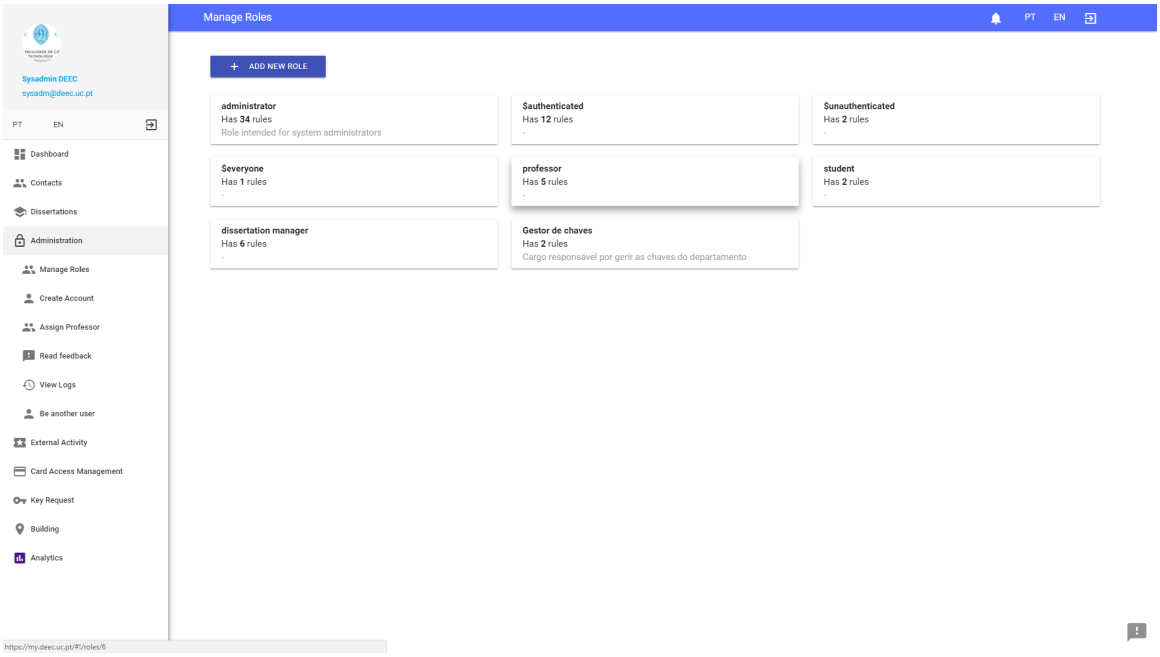


Figure 42: Authorization management final screen.

Figure 42 shows the authorization screen where system administrators can change roles information such as rules and users.

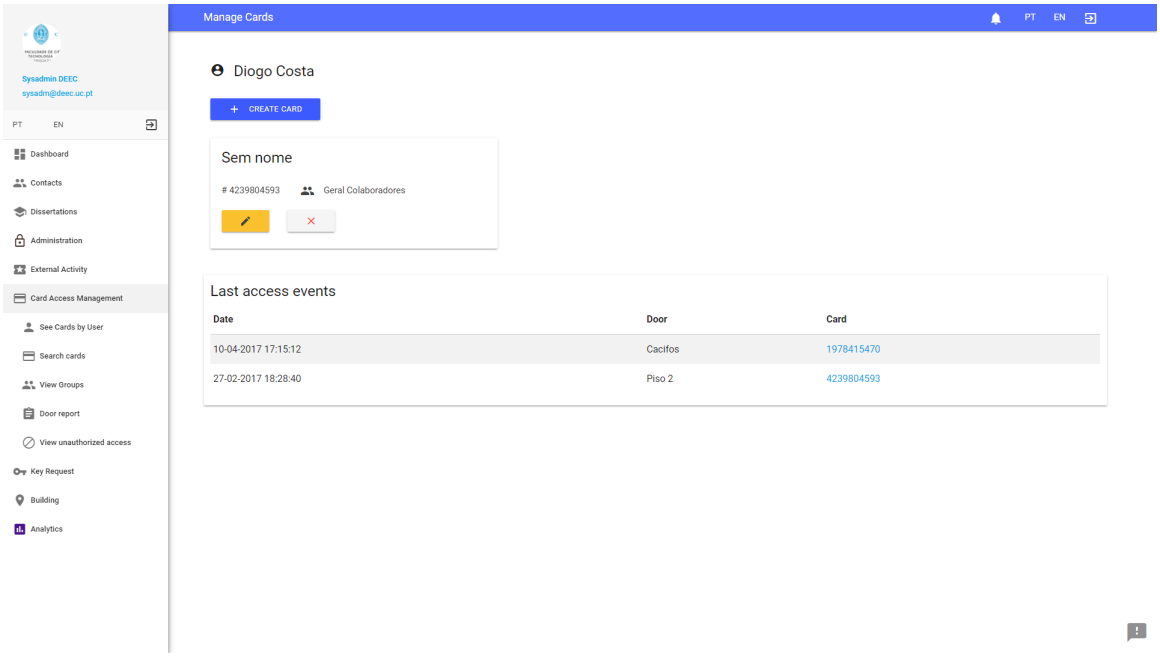


Figure 43: Card management final screen.

Figure 43 shows the screen where card managers can view information about a user. Including user's cards and last events. Here, card managers can also create new cards and delete user cards.

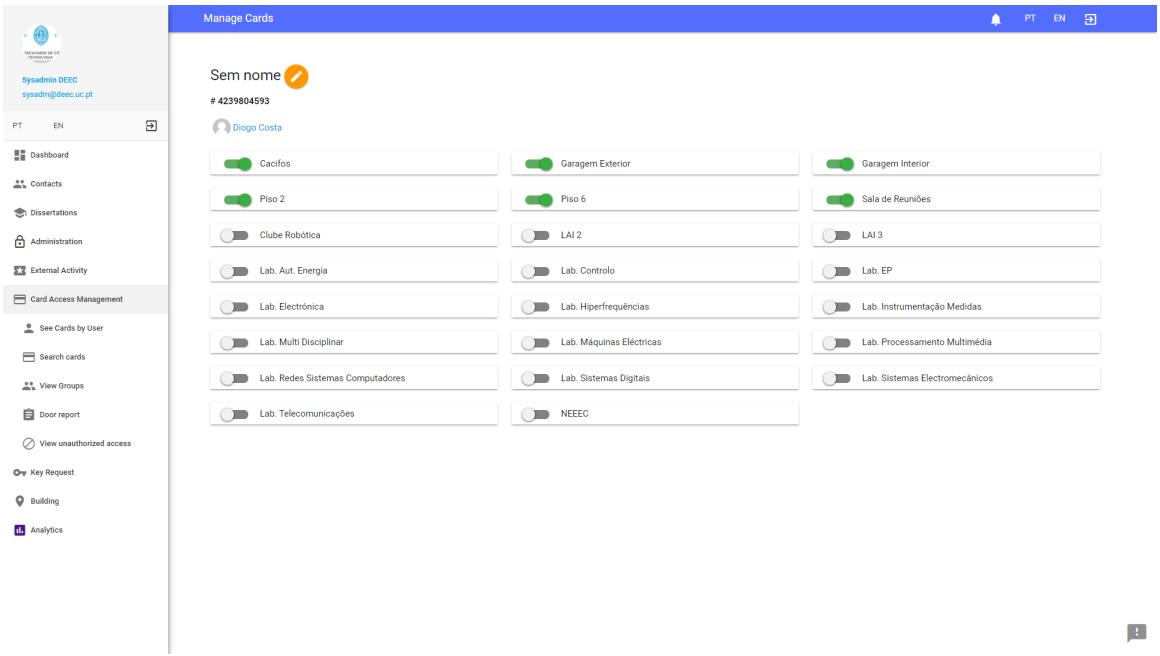


Figure 44: Card rules final screen.

Figure 44 shows the screen where card managers can change rules to a card. Each switch represents a rule that is on or off.

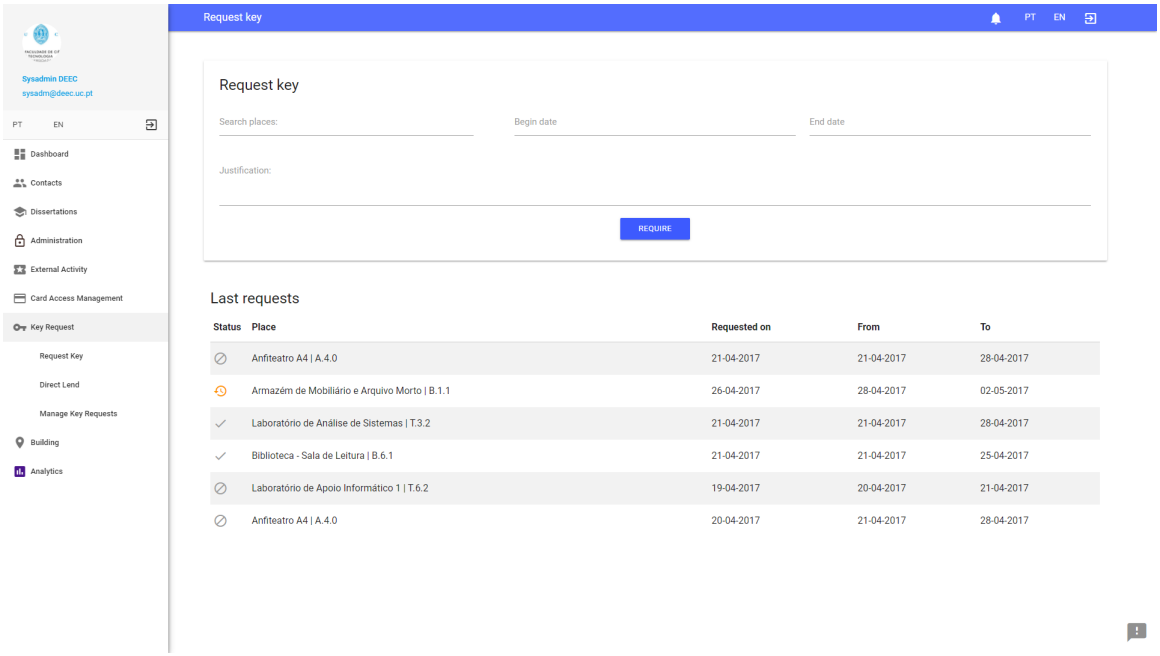


Figure 45: Key request final screen.

Figure 45 shows the screen where users can request a key, they have to choose the room they want to have access, the period when they need the key and justify the request.

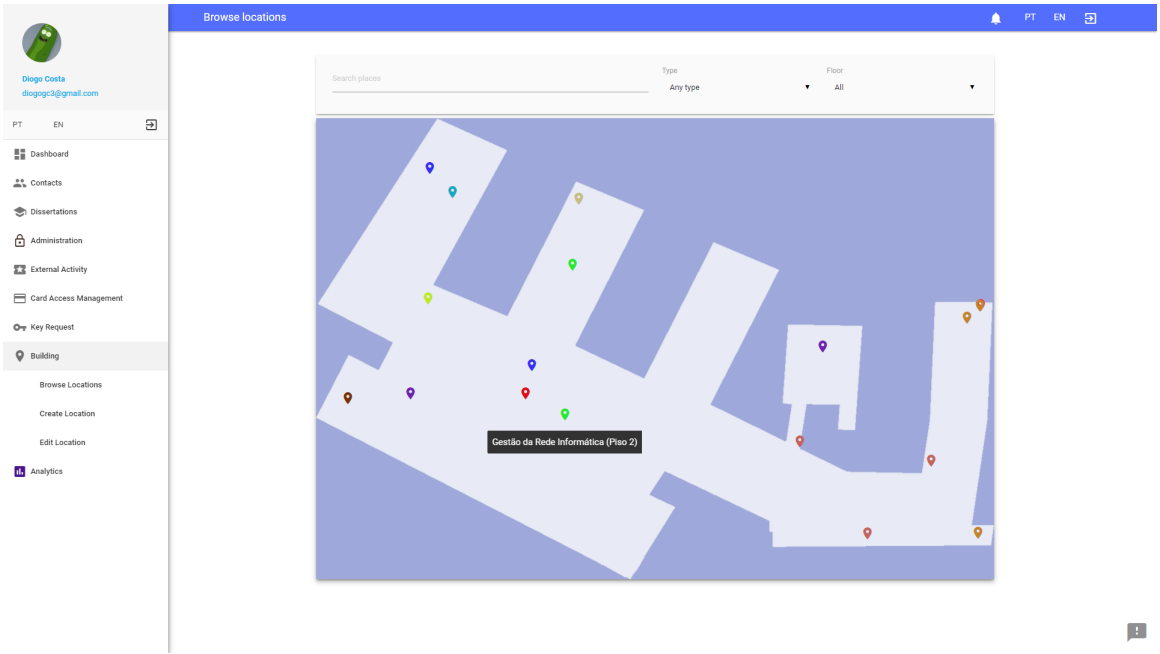


Figure 46: Location browser final screen.

Figure 46 shows the location browser where users can search for places and their location on the department.

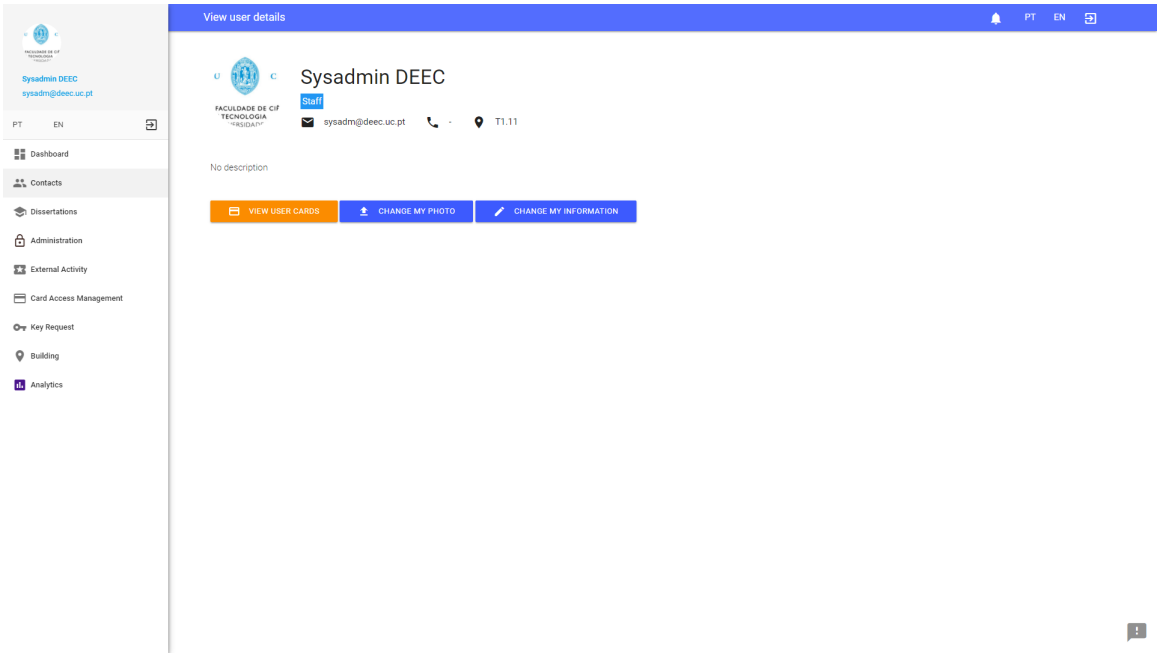


Figure 47: Profile final screen.

Figure 47 shows the profile page that provides information about the user.

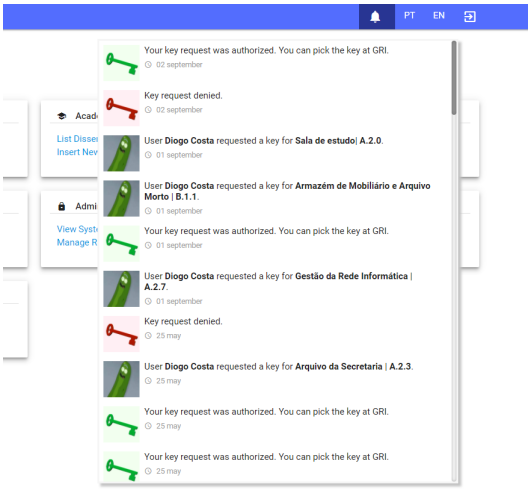


Figure 48: Notifications final screen.

Figure 48 represents a portion of the screen where notifications can be seen. Notifications have representative images, for example they can have an image representing an action or a photo from the user that performed the action that triggered the notification.

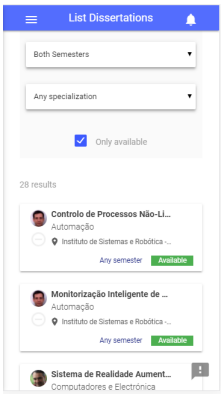


Figure 49: Dissertations final mobile mobile$_d$*issertations*.

Figure 49 shows the dissertations page on a mobile screen. We enhanced the most important pages so they adjust to screen sizes.

8 Testing and results

Having done all the development of all the components, it is time to verify that they match the proposed requirements and actually meets the needs of its users. We will start with the functional testing, then continue with nonfunctional testing (performance, security) and finally we present and discuss the obtained results of the platform usage.

8.1 Functional testing

Due to the scope of the project and the iterative nature of development required to reach our end goals within the proposed timeframe, the initial testing of the web application was performed informally, with several scenarios designed and tested based on the current userbase of the plaform and its expected behaviour. The lack of formal test specification, while in some ways a disadvantage, allowed for faster development, and was put to the test when the application entered production. This transition to production showed that the testing process, although informal, had been successful. The web application was stable and, although the userbase has not yet reached its peak, most issues have been minor (e.g. when exporting assigned dissertations, selecting the semester would not do anything) and fixed quickly due to our error reporting tools (e.g. sentry) and statistics APIs. Taking into account that the application and API are now stable, we are confident that going forward a test suite can be developed to further enhance our test robustness.

To test the API server we used Postman[72] to perform requests. We used it in our development process and think it is also useful for the testing part. It allows us to simulate HTTP requests to the API and analyze the responses. Due to the large amount of test cases we could not automate them. However Postman saves the requests we make so they can be used later. This way, we don't have to always input the test data. In Appendix B we show in detail the tests performed. They were performed along the development process and were recorded so they can be used when needed.

8.2 Usability testing

A formal usability test was not made for the measuring usability of the platform. We believe that for a platform of this size, there should be a formal usability test. But it is hard to gather a significant number of users to test the platform. We instead conducted a series of meetings with professors and department directors to gather feedback and have their perspective about the platform. Because we launched the platform early in the development phase to be used by some users in the department, we could manage to get feedback from those users from those meetings.

- The first meeting was the day *13/04/2017* and served to verify the overall design and if it matched the intended by the direction of the department. The first modules where also demonstrated in detail.
- The second meeting was the day *02/05/2017* focused on the dissertations module and the navigation between pages. For example the suggestion of some *go back* buttons was given there.

- The last meeting was on *19/07/2017* and served to verify if the final platform met the requirements it had and if it was visually uniform. From this meeting, resulted the decision to open the platform to the all users of the department. The platform's divulgation started day *21/07/2017* by sending an email to all the users on the platform.

8.3 Performance testing

To test the performance of our solution we designed a test experience where we vary the number of API nodes in the server and the amount of requests per second. The test setup is shown in figure 50. We used two machines linked to each one with a Gigabit ethernet cable. This ensures the tests are not throttled by the network. Both machines have 4 CPU cores that run at 2.30GHz and 8GB of ram each.

For this performance test, database access is simulated using an in memory database provided by LoopBack. We created a custom method for this test, it checks for authorization and then performs dummy calculations to simulate access to database. Most of the operations in our platform are based on reading data, so this is what we are testing for.

Figure 50: Performance test setup.

We used Apache JMeter[73] to simulate the user requests to the server. Each test has a duration of 1 minute and was run 5 times. In the end the average of the results was calculated and are presented in the following tables. We choose to vary the number of nodes of the API because they are the main component in our architecture. Because our nodes are basically Node.js applications and are single threaded, we chose to test for 1, 2, 4 and 8 nodes. The nodes are in a cluster and are load balanced by PM2 like in our architecture. We also vary the number of requests per second. We started by testing with a small amount of requests per second and incremented them to test the maximum amount of requests our API server can support. We started by 100 requests per second, then tested 500, 1000 and 5000 requests per second. The last number of requests tested was 10000, that was when the application started to produce errors. Because this is a local network, there is no network delay, hence the small values for the average time requests take.

In Tables 8.3, 8.3 and 8.3 we can see the results for 100, 500 and 1000 requests per second. The delay remains low for all these requests. We have to consider that the database is in memory so there is no latency of transferring data and there is almost 0 latency between the two machines. In a production scenario, there is an inherent latency when accessing the database. If we consider that there are on average 3 database accesses (check authentication, check authorization, perform the intended action) and that each request takes approximately 5ms (based on real usage of the production database) to be processed, we can conclude that an average request should take about 17 ms ($5 \times 3 = 15$

ms database latency, 2 ms service operation) and add it to the client-server latency. This value fits our goals for MyDEEC, 1000 requests per second are equivalent to 333 concurrent users performing 3 requests per second, a scenario far beyond our expected usage.

Name	# Nodes	Average time (ms)	Deviation (ms)	% Error	Throughput (req/s)
1.1	1	1	1.5	0	100.8
1.2	2	1	1.55	0	100.8
1.3	4	1	1.57	0	100.8
1.4	8	1	2.25	0	100.7

Table 16: Performance test results (100 requests per second)

Name	# Nodes	Average time(ms)	Deviation (ms)	% Error	Throughput (req/s)
2.1	1	2	7.52	0	497
2.2	2	1	1.8	0	498
2.3	4	2	4.17	0	497
2.4	8	2	3.23	0	498

Table 17: Performance test results (500 requests per second)

Name	# Nodes	Average time(ms)	Deviation (ms)	% Error	Throughput (req/s)
3.1	1	4	2.43	0	988
3.2	2	3	2.33	0	990
3.3	4	6	5.15	0	987
3.4	8	7	3.62	0	989

Table 18: Performance test results (1000 requests per second)

Name	# Nodes	Average time(ms)	Deviation (ms)	% Error	Throughput (req/s)
4.1	1	66	133.09	28.54	4441
4.2	2	9	11.43	0	4574
4.3	4	12	35.7	0	4561
4.4	8	23	47.14	0	4461

Table 19: Performance test results (5000 requests per second)

Name	# Nodes	Average time(ms)	Deviation (ms)	% Error	Throughput (req/s)
5.1	1	80	181.84	45.92	6161
5.2	2	54	99.44	14.77	6046
5.3	4	98	128.84	2.76	6378
5.4	8	309	500.1	48.9	2267

Table 20: Performance test results (10000 requests per second)

In Tables 8.3, 8.3 we can clearly see that the latency increased as it is expected. Errors started to happen at 10000 requests per second which means that it cannot handle that amount of requests. The server CPU was running at 100% during the 10000 requests per seconds experiment. We think that for large amounts of requests, it is better to have 4 nodes running at the same time, it has a higher latency but the percentage of errors is lower.

8.4 Availability testing

To test the availability of our platform, we tested two different scenarios:

- Failure of API nodes - Failures may occur at any time. This scenario simulates various failures on the API nodes and measures errors in normal requests.
- Reload of the entire API cluster - When we need to redeploy the API server we have to tell **PM2** to reload our API server. This scenario simulates a redeploy and measusures errors in normal requests happening at that time.

For both tests, we used a cluster of 4 nodes, this was based on the performance tests presented before.

Failure of API nodes

We created two special testing endpoints for this scenario:

- **/api/availability/get/** - This is a regular endpoint that makes dummy calculations and responds with an HTTP 200 status.
- **/api/availability/failure/** - This endpoint triggers a process exit signal that abruptly terminates the node.

We modeled a traffic source in JMeter that simulates a normal usage, by accessing the */api/availability/get/* endpoint (500 requests per second). We also modeled a traffic source that makes requests to the **/api/availability/failure/** endpoint and causes a failure. These requests occur in intervals of 5 to 15 seconds. We run the test 10 times for 2 minutes each. The uptime of the cluster was 100%, there was always a node to respond to requests, but there was a small percentage (0,02%) of request errors. The reason for this is that nodes would automatically shutdown and requests being processed would produce an error.

Reload of the entire API Server

This test is similar to the previous one. But this time we are directly telling the system to reload all the nodes. This reload is managed by **PM2** and as we saw in the last chapter it reloads it gracefully to ensure there is no downtime. This time we only modeled a traffic source that simulates normal usage by accessing the */api/availability/get/* endpoint, also performing 500 requests per second.

There was 100% uptime, The percentage of errors obtained indicates that there is no downtime when reloading the API server.

8.5 Security testing

Because this is a platform that will be used by the public, it is our duty to verify its security and that it will not be compromised at any point. We studied the top vulnerabilities in the the OWASP TOP 10 Project [52] and then, how we could test them in our platform. We will explain the vulnerabilities tested, how we tested and the results obtained. Unfortunately we did not use automation tools for this task. We knew that for the technologies we used (LoopBack and AngularJS) we would spend to much time integrating the tools and then interpreting the data. So, for the most part where we would inject code, we injected it manually, testing all the places were users can input data like input fields and url queries.

We know that the tests performed by us are simple. We believe that for testing a system like this, a team of professionals is a must, as they try to breach the security of the application at any means with tools and experience acquired through several years of work.

SQL Injection

This type of vulnerability is already covered by the LoopBack framework. But nonetheless we tested each point where users can insert text. As we expected, no SQL could be injected, thus the database cannot be changed nor users can read them by inserting SQL queries in the application.

Cross Site Scripting

For this vulnerability we discovered an issue where JavaScript code could be run on the web application. The feature in question was the log viewing feature. For example, if anyone would insert a dissertation with JavaScript code in the title, it would run when an administrator was viewing the logs. This was a high risk vulnerability that we investigated as soon as we discovered and identified that the problem was due to a library used to highlight search words. The library was removed from the project (thus removing the highlight functionality) but the issue was solved.

Cross Site Request Forgery

To test this vulnerability we only needed to try to make requests (GET, POST) to our API from another web page. Because the API checks the HTTP headers for the access token that identifies the user and the way we manage authorization is by sending that access token in the HTTP headers, pages from other websites cannot simulate an authorized access to our API. The access token is only accessible by pages in the domain *my.deec.uc.pt*, that is our web application. So only our web application can access that token and send it in the HTTP headers.

Broken Access Control And Session Management

As said before, we use access tokens for authentication. They are managed by the LoopBack Framework. Sessions are initialized by authenticating in the LDAP server, this server is only available for the server components, so it is protected from external attacks. All user credentials are sent by secure connections (HTTPS) and no passwords are stored in our server, so attackers cannot have access to them. We performed the following tests:

- Remote access to the LDAP server - We attempted to connect to the LDAP server from outside the server network. We could not manage to access it, the department's firewall blocks outside connections.
- Use an nonexistent access token to access the API and Notifications Server. In both cases the server rejected the request. The API gave HTTP status 401 (Unauthorized) and the Notifications Server responded with a NOT OK status.
- Use an expired access token to access the API and Notifications Server. As the previous test, both servers rejected the request and deleted the expired access token from the database.

Missing Function Level Access Control The base access control (authorization) is provided by the LoopBack framework trough the use of ACLs. But we also have to consider the dynamic authorization feature we implemented. We expect that when rules change, they are applied automatically as soon as they reach the API.

8.6 Platform usage and statistics

In the past chapter we mentioned the analytics feature we developed. Unfortunately we only had the idea of developing the feature in the end of July, so we only have data from the month of August. According with Figure 51, during this month, we had 63 unique users that made a total of 151 sessions and viewed 1876 pages. This is is not much, but we should note that during this month, most of the people are in holidays and do not

perform academic activity. Hence, we think these are really good values and we think that during the month of September (when the new academic year starts) usage of the platform will tend to increase.

Figure 51: Global statistics for the web application, taken from the analytics screen.

In Figure 52, usage per hour is shown. We can clearly see that the period comprehended between 9h and 18h is where most of the activity takes place, afternoon being the most active part of the day. It makes sense, as these are the active hours of the day.

When we analyzed this information, the activity at midnight was high. It was strange and we had to look up the real recorded events to discover if it was an error causing that amount of views at midnight. Those views actually correspond to users and are valid, we even checked if the view times per page made sense. It is really strange due to the trend along the day, but are valid views.

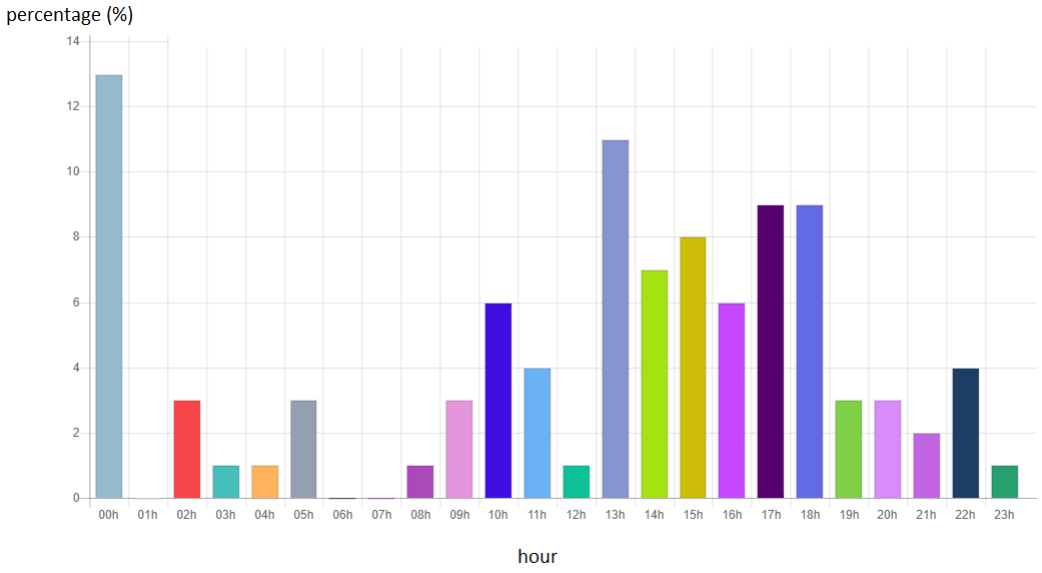


Figure 52: Percentage of view per hour of the day, taken from the analytics screen.

Knowing what pages users view is important to have notion of what is really happening in the platform. Figure 53 shows the views per page. The login page is not shown here, we only record this information for authenticated users. Dashboard being the first page a user sees when entering the platform, makes it being the most viewed page, followed by the dissertations listing and viewing, and the contacts. Other features tend to be less used in this month, but for example the card management module was strongly used during the other months, when people wanted to created cards. It will sure continue to be used when the new year starts.

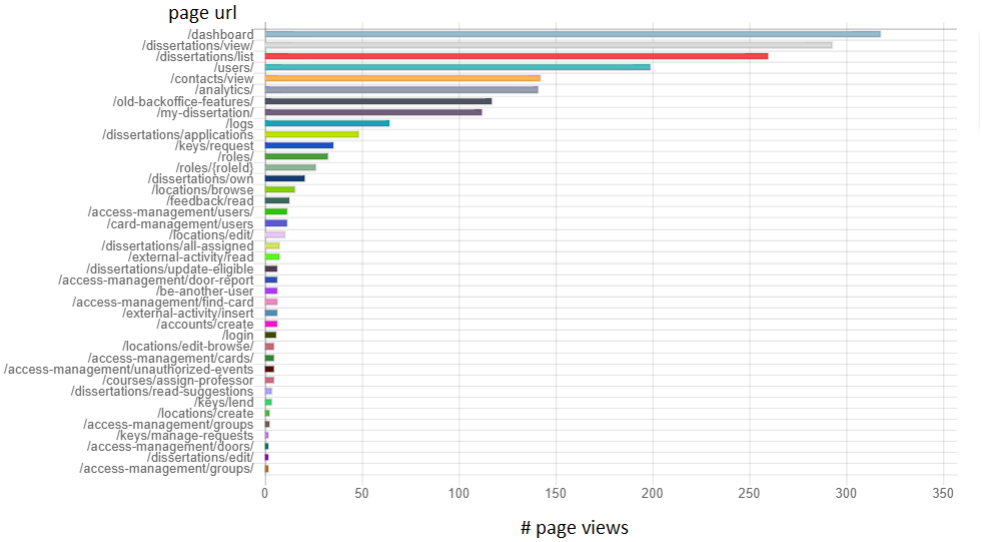


Figure 53: Amount of views per page, taken from the analytics screen.

9 Conclusions

Reaching the end of this document, we can say that the goals were met. We have a stable platform that is being used right now for users of the entire DEEC department. We hope that more users will start to use the platform as soon as the new academic year starts. And that the future developers of the platform can take advantage of all the work done to make the development process easier.

The implementation part was the process that consumed the most time on this project. From the extensions we had to do to the LoopBack framework and the optimization we had to make to have a better application, it all was worth it. We feel that maybe more time could have been invested in testing the platform. But nevertheless, we think that we have a stable platform. And now, the functional tests presented in this document can be formalized to ensure that in the future, testing will be more easier and simple.

Personally, I am happy that the goals were met. Of course some things like automated testing were not implemented. But working so many hours on this project and having it actually being used by people in the department brings me a sensation of accomplishment. I learned so many new things, not only on the technical level, but also on the social level. and that is all that knowledge that really matters.

References

[1] StrongLoop. LoopBack. http://loopback.io/. (Last access date: 12/01/2017).

[2] DreamHost. Web server performance comparison. https://help.dreamhost.com/hc/en-us/articles/215945987-Web-server-performance-comparison. (Last access date: 02/01/2017).

[3] Built With. Web servers usage statistics. https://trends.builtwith.com/web-server. (Last access date: 02/08/2017).

[4] Streamline. Streamline. http://streamline.pt/. (Last access date: 19/01/2017).

[5] Streamline. Deec backoffice. https://backoffice.deec.uc.pt/. (Last access date: 15/01/2017).

[6] W3Schools. Responsive web design. http://www.w3schools.com/html/html_responsive.asp. (Last access date: 20/01/2017).

[7] T. Berners-Lee. World wide web rfc. https://www.ietf.org/rfc/rfc1630.txt, 1994. (Last access date: 14/09/2016).

[8] T. Berners-Lee. Hypertext markup language. https://tools.ietf.org/html/rfc1866, 1995. (Last access date: 17/01/2017).

[9] W3 schools. Html tags. http://www.w3schools.com/tags/. (Last access date: 18/01/2017).

[10] Oracle. Java. https://www.java.com. (Last access date: 22/09/2016).

[11] The PHP Group. Php. https://secure.php.net/. (Last access date: 22/09/2016).

[12] Network Working Group. The text/css media type. https://tools.ietf.org/html/rfc2318, 1998. (Last access date: 30/09/2016).

[13] Network Working Group. The text/css media type. https://tools.ietf.org/html/rfc4329, 2006. (Last access date: 29/09/2016).

[14] E. Rescorla. Http over tls. https://tools.ietf.org/html/rfc2818, 2000. (Last access date: 20/09/2016).

[15] Mozilla Developer Network. Html5 -web developer guides. https://developer.mozilla.org/en-US/docs/Web/Guide/HTML/HTML5. (Last access date: 10/01/2017).

[16] Node.js Foundation. Node.js. https://nodejs.org. 12/01/2017.

[17] AngularJS team. Angularjs. https://angularjs.org/. (Last access date: 18/01/2017).

[18] Google. Chrome V8. https://developers.google.com/v8/. (Last access date: 26/12/2016).

[19] W3Techs. Usage statistics and market share of node.js for websites. https://w3techs.com/technologies/details/ws-nodejs/all/all, 2017. (Last access date: 18/01/2017).

[20] StrongLoop. Why Node.js? https://strongloop.com/node-js/why-node/. (Last access date: 28/09/2016).

[21] Todd Hoff. LinkedIn Moved From Rails To Node: 27 Servers Cut And Up To 20x Faster. http://highscalability.com/blog/2012/10/4/linkedin-moved-from-rails-to-node-27-servers-cut-and-up-to-2.html. (Last access date: 28/09/2016).

[22] npm. npm. https://www.npmjs.com/. (Last access date: 11/01/2017).

[23] Erik DeBill. Module counts. http://www.modulecounts.com/, 2013. (Last access date: 18/01/2017).

[24] IBM. Ibm acquires strongloop to extend enterprise reach using ibm cloud. https://www-03.ibm.com/press/us/en/pressrelease/47577.wss, 2015. (Last access date: 29/09/2016).

[25] StrongLoop. LoopBack - Controlling data access. http://loopback.io/doc/en/lb2/Controlling-data-access.html. (Last access date: 17/11/2016).

[26] Express. Express. http://expressjs.com/. (Last access date: 27/12/2016).

[27] Express. Using middleware. http://expressjs.com/en/guide/using-middleware.html. (Last access date: 29/08/2016).

[28] AngularJS. Using $location. https://docs.angularjs.org/guide/$location, 2017. (Last access date: 30/09/2016).

[29] Less Team. Less. http://lesscss.org/, 2009. (Last access date: 19/10/2016).

[30] Sass team. Sass. http://sass-lang.com/, 2006. (Last access date: 19/10/2016).

[31] Bootstrap Core Team. Boostrap. https://getbootstrap.com/. (Last access date: 12/10/2016).

[32] ZURB. Foundation. http://foundation.zurb.com/. (Last access date: 13/10/2016).

[33] Materialize. Materialize. http://materializecss.com/. (Last access date: 19/01/2017).

[34] Google. Material design. https://material.google.com. (Last access date: 14/10/2016).

[35] Inc. A. Melnikov Isode Ltd. I. Fette, Google. The websocket protocol. https://tools.ietf.org/html/rfc6455, 2011. (Last access date: 20/10/2016).

[36] Socket.io. Socket.io. http://socket.io/. (Last access date: 20/10/2016).

[37] Node.js. Node.js http module. https://nodejs.org/api/http.html. (Last access date: 30/09/2016).

[38] Apache Software Foundation. Apache http server. https://httpd.apache.org/. (Last access date: 27/12/2016).

[39] NGINX Inc. Nginx. https://www.nginx.com/. (Last access date: 16/01/2017).

[40] Jesse Storimer. How many threads is too many? http://www.jstorimer.com/blogs/workingwithcode/7970125-how-many-threads-is-too-many, 2013. (Last access date: 03/01/2017).

[41] Remy Sharp. Nodemon. https://nodemon.io/. (Last access date: 25/08/2017).

[42] OStrongLoop. Strongloop process manager. http://strong-pm.io/. (Last access date: 25/08/2017).

[43] PM2 Team. Pm2 - advanced, production process manager for node.js. http://pm2.keymetrics.io/. (Last access date: 03/01/2017).

[44] Universidade de Coimbra. Inforestudante. https://inforestudante.uc.pt. (Last access date: 19/01/2017).

[45] The CentOS Project. Centos. https://www.centos.org/. (Last access date: 25/08/2017).

[46] Oracle Corporation. Mysql. https://www.mysql.com/. (Last access date: 15/12/2016).

[47] Open Web Application Security Project. Injection vulnerability. https://www.owasp.org/index.php/Top_10_2013-A1-Injection. (Last access date: 20/08/2017).

[48] Open Web Application Security Project. Cross-site scripting (xss) vulnerability. https://www.owasp.org/index.php/Top_10_2013-A3-Cross-Site_Scripting_(XSS). (Last access date: 20/08/2017).

[49] Technovelgy. Radio-frequency identification. http://www.technovelgy.com/ct/technology-article.asp. (Last access date: 15/11/2016).

[50] Microsoft. Microsoft windows operating systems. https://www.microsoft.com/en-us/windows/. (Last access date: 10/08/2017).

[51] DSDM Consortium. Moscow prioritisation. https://www.agilebusiness.org/content/moscow-prioritisation-0. (Last access date: 20/11/2016).

[52] Open Web Application Security Project. Owasp top 10 project. https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project. (Last access date: 20/08/2017).

[53] Open Web Application Security Project. Cross-site request forgery (csrf) vulnerability. https://www.owasp.org/index.php/Top_10_2013-A8-Cross-Site_Request_Forgery_(CSRF). (Last access date: 20/08/2017).

[54] Open Web Application Security Project. Broken authentication and session management vulnerability. https://www.owasp.org/index.php/Top_10_2013-A2-Broken_Authentication_and_Session_Management. (Last access date: 20/08/2017).

[55] Open Web Application Security Project. Missing function level access control vulnerability. https://www.owasp.org/index.php/Top_10_2013-A7-Missing_Function_Level_Access_Control. (Last access date: 20/08/2017).

[56] Simon Brown. C4 model. http://www.codingthearchitecture.com/2014/08/24/c4_model_poster.html. (Last access date: 9/08/2017).

[57] Inc. Functional Software. Sentry. https://sentry.io. (Last access date: 25/08/2017).

[58] Nodemailer. Nodemailer. https://nodemailer.com/. (Last access date: 05/01/2017).

[59] W3Schools.com. Html iso-8859-1 reference. https://www.w3schools.com/charsets/ref_html_8859.asp. (Last access date: 20/08/2017).

[60] W3Schools.com. Html unicode (utf-8) reference. https://www.w3schools.com/charsets/ref_html_utf8.asp. (Last access date: 20/08/2017).

[61] José Ribeiro Witesmith. Mysql encoding hell: How to export utf-8 data from a latin1 table. https://www.whitesmith.co/blog/latin1-to-utf8/. (Last access date: 12/08/2017).

[62] Microsoft. Sql server technical documentation. https://docs.microsoft.com/en-us/sql/sql-server/sql-server-technical-documentation. (Last access date: 9/08/2017).

[63] Ldap Passport Developers. Ldap passport. https://github.com/vesse/passport-ldapauth. (Last access date: 4/08/2017).

[64] Ldapjs. Ldapjs. http://ldapjs.org/. (Last access date: 9/08/2017).

[65] kangax. Ecmascript compatibility table. https://kangax.github.io/compat-table/es6/. (Last access date: 19/08/2017).

[66] Babel team. Babel, javascript compiler. https://babeljs.io/. (Last access date: 19/08/2017).

[67] Webpack team. Webpack module bundler. https://webpack.js.org/. (Last access date: 19/08/2017).

[68] Oren Ben-Kiki Clark C. Evans, Ingy döt Net. Yaml. http://yaml.org/. (Last access date: 19/08/2017).

[69] Mozilla Developer Network (MDN). Javascript regexp. https://developer.mozilla.org/en/docs/Web/JavaScript/Guide/Regular_Expressions. (Last access date: 20/08/2017).

[70] M. Mealling Refactored Networks R. Salz DataPower Technology P. Leach, Microsoft. Common format and mime type for comma-separated values (csv) files. https://tools.ietf.org/html/rfc4122, 2005. (Last access date:20/08/2017).

[71] Chris Bateman. Webpack visualizer. https://chrisbateman.github.io. (Last access date: 12/08/2017).

[72] Inc Postdot Technologies. Postman. https://www.getpostman.com/. (Last access date: 25/08/2017).

[73] Apache Software Foundation. Apache jmeter. https://jmeter.apache.org/. (Last access date: 25/08/2017).

[74] Inc. Y. Shafranovich, SolidMatrix Technologies. Common format and mime type for comma-separated values (csv) files. https://tools.ietf.org/html/rfc4180, 2005. (Last access date:01/08/2017).

Appendices

A Detailed requirements

This appendix describes in detail the full requirements of our the proposed platform.

01.01 Login	
Primary Actors	Unauthenticated user
Secondary Actors	-
Description	Users with a DEEC email must be able to login on the platform using their email and password.
Flow	1. Fill the user email using either @alunos.deec.uc.pt or @deec.uc.pt email 2. Fill the password 3. Click the submit button
Pre-conditions	User is not authenticated. User is on the login page. The login details inserted are valid.
Post-conditions	User becomes authenticated.
Priority	Must have
Specificity	Sea level (user goal)
Source	Authentication

Table 21: Use case specification 01.01 - Login

01.02 Logout	
Primary Actors	Authenticated user
Secondary Actors	-
Description	Authenticated users must be able to logout from any application screen.
Flow	1. Click the logout button
Pre-conditions	User is authenticated. User is on any page
Post-conditions	User becomes unauthenticated
Priority	Must have
Specificity	Sea level (user goal)
Source	Authentication

Table 22: Use case specification 01.02 - Logout

02.01 Assign user to a role	
Primary Actors	System Administrators
Secondary Actors	Any user
Description	System administrators must be able to assign a person to a certain role. Users must be able to have more than one role.
Flow	1. Open the desired role from the role list 2. Click the button to add new user 3. Type the user name 4. Click the user
Pre-conditions	There is at least one role available.
Post-conditions	Role is assigned to the user.
Priority	Must have
Specificity	Sea level (user goal)
Source	Authorization

Table 23: Use case specification 02.01 - Assign user to a role

02.02 Create role	
Primary Actors	System administrators
Secondary Actors	-
Description	System administrators must be able to create a new role in the platform.
Flow	1. Click the **add role** button in role list 2. Fill the role name 3. (Optional) Fill the role description 4. Click the **create** button
Pre-conditions	The name of the role does not exist.
Post-conditions	Role is created
Priority	Should have
Specificity	Sea level (user goal)
Source	Authorization

Table 24: Use case specification 02.02 - Create role

02.03 Delete role	
Primary Actors	System administrators
Secondary Actors	-
Description	System administrators must be able to delete roles previously created.
Flow	1. Open the role details in the role list 2. Click the **delete** button 3. Accept the confirmation dialog
Pre-conditions	The role was previously created and is not a default role from the platform.
Post-conditions	Role is deleted.
Priority	Should have
Specificity	Sea level (user goal)
Source	Authorization

Table 25: Use case specification 02.03 - Delete role

02.04 Change role permissions	
Primary Actors	System administrators
Secondary Actors	-
Description	System administrators must be able to change role permissions. This includes the creation or deletion of rules from a role. Those rules dictate if that role can access a certain feature of the platform.
Flow	1. Choose the role from the list 2. (optional) Choose the features that must be allowed to that particular role from the list 3. (optional) Remove permissions from the role list 4. Permissions are automatically saved
Pre-conditions	There is at least one role on the platform.
Post-conditions	The permissions for that role are modified.
Priority	Should have
Specificity	Sea level (user goal)
Source	Authorization

Table 26: Use case specification 02.04 - Change role permissions

03.01 List dissertations	
Primary Actors	Authenticated users
Secondary Actors	-
Description	Users must be able to list all the dissertations in the platform. Dissertations can be filtered by active status (available for application, in progress, concluded), course specialization, semester and title search. Results for dissertations are paginated, having a certain amount of them per page.
Flow	1. Be in the dissertation list page 2. Dissertations are automatically shown 3. (optional) Search by name/content 4. (optional) Filter by semester 5. (optional) Filter by specialization 6. (optional) Filter by status (available or not)
Pre-conditions	There is at least one dissertation on the platform.
Post-conditions	-
Priority	Must have
Specificity	Sea level (user goal)
Source	Dissertations

Table 27: Use case specification 03.01 - List dissertations

03.02 View dissertation details	
Primary Actors	Authenticated users
Secondary Actors	-
Description	Users must be able to view the details of the dissertations, including all the relevant information: • Title • Supervisors (internal and external) • Status (available, in progress, finished, abandoned) • Assigned student (if applicable) • Description • Remarks • Work place • Specialization • Semester
Flow	1. The user sees the details
Pre-conditions	The dissertation exists
Post-conditions	-
Priority	Must have
Specificity	Sea level (user goal)
Source	Dissertations

Table 28: Use case specification 03.02 - View dissertation details

03.03 Insert/edit dissertation	
Primary Actors	Professors
Secondary Actors	Student
Description	Professors must be able to insert/edit a new dissertation in the platform. The platform must enable the insertion of the Portuguese and English information. Furthermore, a student can be assigned to the dissertation by the professor when inserting it. Dissertations do not have an assigned semester, the semester is automatically filled when a student is selected.
Flow	1. Insert Portuguese title 2. Insert Portuguese description 3. Insert English title 4. Insert Portuguese title 5. Choose the specialization 6. Insert the work place 7. (optional) Add associated internal supervisors from the list 8. (optional) Add associated external supervisors information (name, institution, role, email, phone number) 9. (optional) Select a student from the list to whom the dissertation is assigned 10. Click the save button
Pre-conditions	Only eligible students can be assigned.
Post-conditions	Dissertation is created. If a student is assigned, a notification is sent to him/her.
Priority	Must have
Specificity	Sea level (user goal)
Source	Dissertations

Table 29: Use case specification 03.03 - Insert/edit dissertation

03.04 Delete dissertation	
Primary Actors	Professors/Dissertation managers
Secondary Actors	-
Description	Professors must be able to delete their own dissertations. Dissertation managers can delete any dissertation.
Flow	1. Click the **remove** button on dissertation details 2. Accept the confirmation dialog box
Pre-conditions	The dissertation belongs to the professor.
Post-conditions	Dissertation is deleted from the platform.
Priority	Must have
Specificity	Sea level (user goal)
Source	Dissertations

Table 30: Use case specification 03.04 - Delete dissertation

03.05 Toggle dissertation visibility	
Primary Actors	Professors/Dissertation managers
Secondary Actors	-
Description	Professors must be able to toggle the visibility of their own dissertations. Invisible dissertations do not appear on the dissertations list. Dissertation managers can toggle any dissertation.
Flow	1. Click the **toggle visibility** button on dissertation details
Pre-conditions	The dissertation belongs to the professor.
Post-conditions	Dissertation visibility is changed.
Priority	Should have
Specificity	Sea level (user goal)
Source	Dissertations

Table 31: Use case specification 03.05 - Toggle dissertation visibility

03.06 Clone dissertation	
Primary Actors	Professors/Dissertation managers
Secondary Actors	-
Description	Professors must be able to clone their own dissertations. Dissertation managers can clone any dissertation.
Flow	1. Click the **clone dissertation** button on dissertation details 2. Accept the confirmation dialogue
Pre-conditions	The dissertation belongs to the professor.
Post-conditions	A new dissertation is created with all the field of the cloned one.
Priority	Should have
Specificity	Sea level (user goal)
Source	Dissertations

Table 32: Use case specification 03.06 - Clone dissertation

03.07 Apply to dissertation	
Primary Actors	Students
Secondary Actors	Professor
Description	Eligible students must be able to apply to an available dissertation.
Flow	1. Click the **apply** button on any available dissertation details 2. Accept the confirmation dialog box
Pre-conditions	The student is eligible for application.
Post-conditions	Student is applied to the chosen dissertation. A notification is sent ft the professor.
Priority	Must have
Specificity	Sea level (user goal)
Source	Dissertations

Table 33: Use case specification 03.07 - Apply to dissertation

03.08 Accept applied student	
Primary Actors	Professors / Dissertation Managers
Secondary Actors	Student
Description	Professors must be able to accept students that apply to their own dissertation. Dissertation managers must be able to accept students that apply to all dissertations. When a student is accepted, the status of the dissertation is automatically changed to ”in progress” and can no longer accept applications from the students.
Flow	1. Be on the dissertation details 2. Click the assign button on the student application 3. Click the confirm button on the dialog box
Pre-conditions	The professor is the owner of the dissertation. There is at least one student applied to the dissertation.
Post-conditions	The student is assigned to the dissertation. A notification is sent to the student.
Priority	Must have
Specificity	Sea level (user goal)
Source	Dissertations

Table 34: Use case specification 03.08 - Accept applied student

03.09 Update eligible students	
Primary Actors	Dissertation managers
Secondary Actors	Student
Description	The platform must enable the upload of a file containing the eligible students of each semester. This file is in CSV format[74]. There must be a template CSV file that dissertation managers can download to check the required fields.
Flow	1. Click the upload button 2. Select the CSV file containing the eligible students 3. Click the submit button
Pre-conditions	Valid CSV file
Post-conditions	Students in the file become eligible.
Priority	Must have
Specificity	Sea level (user goal)
Source	Dissertations

Table 35: Use case specification 03.09 - Update eligible students

03.10 List applications	
Primary Actors	Students
Secondary Actors	-
Description	Students must be able to list dissertations they have applied to and remove that application.
Flow	1. Click the list applications button on the navigation bar 2. (optional) Click the remove application on a dissertation 3. (optional) Click accept on the confirmation dialog box
Pre-conditions	The student has at least on application to a dissertation
Post-conditions	Application to the dissertation is deleted (if applicable).
Priority	Must have
Specificity	Sea level (user goal)
Source	Dissertations

Table 36: Use case specification 03.10 - List applications

03.11 List owned dissertations	
Primary Actors	Professors
Secondary Actors	-
Description	Professors must be able to view their inserted dissertations including their status.
Flow	1. Click the **list owned dissertations** button on the navigation bar
Pre-conditions	The professor has at least one dissertation.
Post-conditions	-
Priority	Must have
Specificity	Sea level (user goal)
Source	Dissertations

Table 37: Use case specification 03.11 - List owned dissertations

03.12 View assigned dissertation	
Primary Actors	Student
Secondary Actors	-
Description	Students must be able to view the dissertations they are assigned to.
Flow	1. Click on the dissertation view menu
Pre-conditions	The student has a dissertation assigned.
Post-conditions	-
Priority	Must have
Specificity	Sea level (user goal)
Source	Dissertations

Table 38: Use case specification 03.12 - View assigned dissertation

03.13 Submit dissertation suggestion	
Primary Actors	Students
Secondary Actors	-
Description	Students must be able to submit dissertation suggestions.
Flow	1. Fill the **suggestion text area** 2. Click the **submit** button
Pre-conditions	-
Post-conditions	Suggestion is saved.
Priority	Should have
Specificity	Sea level (user goal)
Source	Dissertations

Table 39: Use case specification 03.13 - Submit dissertation suggestion

03.14 Read dissertation suggestions	
Primary Actors	Professors/Dissertation managers
Secondary Actors	-
Description	Professors and dissertation managers must be able to read suggestions submited by the students.
Flow	1. Click the **view submited suggestions** in the menu 2. (Optional) Select a suggestions to view its details (suggestion, student details, date)
Pre-conditions	-
Post-conditions	-
Priority	Should have
Specificity	Sea level (user goal)
Source	Dissertations

Table 40: Use case specification 03.14 - Read dissertation suggestions

04.01 Manage card access groups	
Primary Actors	System Administrators
Secondary Actors	-
Description	The platform must enable the creation and removal of access groups. For each group, rules can be created or removed. Those rules include the granted doors for that particular group. The card access management page has a list of the groups available.
Flow	This flow is divided into 2 steps: • Create access group 1. Click the **create access group** button 2. Insert group name 3. Click **save** button • Add/remove rule to access group 1. Click the **manage rules** button on the access group 2. (optional) Select doors which are accessible to the group 3. (optional) Remove accessible door to the group 4. Click the **save changes** button
Pre-conditions	Be at the card access management page.
Post-conditions	Changes are saved
Priority	Can have
Specificity	Sea level (user goal)
Source	Card access management

Table 41: Use case specification 04.01 - Manage card access groups

04.02 Edit card rules	
Primary Actors	System Administrators
Secondary Actors	Any user
Description	The platform must enable the possibility of changing rules associated with a user's card. This includes the granting or denying of access to doors for a certain card. Those rules override the ones from the access group the card is associated with. Changes are automatically saved upon the toggle of any rule.
Flow	1. Select a card from the user profile/card list view 2. (optional) Toggle a rule relative to a certain door (authorize/deny it)
Pre-conditions	Be at a user profile/card list view.
Post-conditions	Custom rules to the user are saved.
Priority	Must have
Specificity	Sea level (user goal)
Source	Card access management

Table 42: Use case specification 04.02 - Edit card rules

04.03 Create a new card	
Primary Actors	System Administrators
Secondary Actors	Any user
Description	The platform must enable the creation of user's cards. Every card is identified by its RFID number and optionally, a name.
Flow	1. Click on the **create new card** button on the user profile 2. Fill the card number 3. (optional) Fill the name 4. Click the **save** button
Pre-conditions	Be at the user profile.
Post-conditions	The new card is created.
Priority	Must have
Specificity	Sea level (user goal)
Source	Card access management

Table 43: Use case specification 04.03 - Create a new card

04.04 Delete user's card	
Primary Actors	System Administrators
Secondary Actors	Any user
Description	System administrators must be able to delete any card.
Flow	1. Click on the **delete card** button on the desired card 2. Accept the confirmation dialog
Pre-conditions	Be at the user profile.
Post-conditions	The card is deleted
Priority	Must have
Specificity	Sea level (user goal)
Source	Card access management

Table 44: Use case specification 04.04 - Delete user's card

04.05 View unauthorized card accesses	
Primary Actors	System Administrators
Secondary Actors	Any user
Description	System administrators must be able view unauthorized card accesses to any door of the department. This list must include the time of event, the door where it occured, the card number, and if possible, the person who triggered that event.
Flow	1. Click on the **view unauthorized accesses** sub menu under card management menu 2. (Optional) Click other pages to load more results
Pre-conditions	-
Post-conditions	-
Priority	Must have
Specificity	Sea level (user goal)
Source	Card access management

Table 45: Use case specification 04.05 - View unauthorized card accesses

04.06 View user's last accesses	
Primary Actors	System Administrators
Secondary Actors	Any user
Description	System administrators must be able to view a list of the last access events for a certain user. This list of events is displayed on the user profile and contains the time of event, the door where it occured and the card used.
Flow	1. Scroll into the **last events** section in the user profile
Pre-conditions	-
Post-conditions	-
Priority	Must have
Specificity	Sea level (user goal)
Source	Card access management

Table 46: Use case specification 04.06 - View user's last accesses

04.07 View door's last accesses	
Primary Actors	System Administrators
Secondary Actors	Any user
Description	System administrators must be able to view a list of the last access events for a every door. This is very similar to the unauthorized card accesses.
Flow	1. Choose the door from the door list 2. (Optional) Click other pages to get more results
Pre-conditions	-
Post-conditions	-
Priority	Must have
Specificity	Sea level (user goal)
Source	Card access management

Table 47: Use case specification 04.07 - View door's last accesses

04.08 View/export list of whom has access to which door	
Primary Actors	System Administrators
Secondary Actors	Any user
Description	System administrators must be able to view a list of whom is authorized to access which door. This is to be represented as a matrix where the rows are users and the columns are the doors. This list must be also exportable.
Flow	1. Click the **door report** sub menu under the card management menu 2. (Optional) Click the **export data** button
Pre-conditions	-
Post-conditions	If that is exported, a CSV file is downloaded containing all the finromation on the page.
Priority	Should have
Specificity	Sea level (user goal)
Source	Card access management

Table 48: Use case specification 04.08 - View/export list of whom has access to which door

05.01 View courses	
Primary Actors	Authenticated users
Secondary Actors	-
Description	All authenticated users must be able to search for courses and view their detailed information. Details shown include: • Course name • Code • Scientific area • Course • Status • Year • Goals • Schedule
Flow	1. Click the **list courses** button on the navigation bar 2. (optional) Search by name 3. (optional) Filter by year 4. (optional) Click the **show details** button
Pre-conditions	There is at least one course.
Post-conditions	-
Priority	Can have
Specificity	Sea level (user goal)
Source	Academic

Table 49: Use case specification 05.01 - View courses

05.02 Manage courses	
Primary Actors	Secretary
Secondary Actors	-
Description	Secretary must be able to manage courses. This includes, creation, deletion and edition.
Flow	The flow is divided into 3 distinct steps: • Create: 1. Click the **create course** button 2. Insert the course name 3. Insert the course code 4. Select the scientific area 5. Select the course 6. Select the status 7. Select the year 8. Insert the goals 9. Insert the schedule 10. Click the **save changes** button • Modify: 1. Click the **edit** button on a course in the list 2. Change any field 3. Click the **save changes** button • Delete: 1. Click the **delete** button on a course in the list 2. Click the **confirm** button in the dialog box
Pre-conditions	The user is on the list courses page.
Post-conditions	Course changes are saved.
Priority	Can have
Specificity	Sea level (user goal)
Source	Academic

Table 50: Use case specification 05.02 - Manage courses

06.01 View personal information	
Primary Actors	Any authenticated user
Secondary Actors	-
Description	Any user must be able to view their and other people's personal information at any time. This includes their email, phone number, photo and biography if applicable. The ideia is that when a name appears anywhere on the platform it is accompanied with the user's photo and links to the user profile.
Flow	1. Click the name or photo when a user is mentioned anywhere on the platform
Pre-conditions	-
Post-conditions	-
Priority	Must have
Specificity	Sea level (user goal)
Source	Information

Table 51: Use case specification 06.01 - View personal information

06.02 View organization contacts	
Primary Actors	Any authenticated user
Secondary Actors	-
Description	A page with important contacts that anyone must be able to view. Contacts include. • Name of the service • Person(s) in charge of the service • Email • Other relevant contacts • Contact photo
Flow	1. Click on the **view contacts** on the navigation bar 2. Click on a contact name to view the details (this links to the view personal information requirement)
Pre-conditions	-
Post-conditions	-
Priority	Must have
Specificity	Sea level (user goal)
Source	Information

Table 52: Use case specification 06.02 - View organization contacts

06.03 View building information	
Primary Actors	Any authenticated user
Secondary Actors	-
Description	A page with the different department places and details about them must be present on the platform.
Flow	1. Click on the **view department information** menu 2. A map of the department appears with the different rooms 3. (optional) Change the floor 4. (optional) Click on a room and relevant information about it is shown
Pre-conditions	-
Post-conditions	-
Priority	Must have
Specificity	Sea level (user goal)
Source	Information

Table 53: Use case specification 06.03 - View building information

06.04 Email notifications	
Primary Actors	Any user
Secondary Actors	-
Description	Any important action affecting a user, must issue an email that will be sent to that user.
Flow	1. An important action that affects a user occurs 2. Email is sent to that user
Pre-conditions	Important action occurs
Post-conditions	Email is sent
Priority	Must have
Specificity	Sea level (user goal)
Source	Information

Table 54: Use case specification 06.04 - Email notifications

06.05 Web notifications	
Primary Actors	Authenticated users
Secondary Actors	-
Description	Any important action affecting a user, must issue a notification that will be sent to that user. If the user is online on the platform, the notification must be delivered instantly, otherwise the notification will be stored and will be sent when the user comes back online.
Flow	1. An important action involving a user occurs 2. Real-time notification is sent to that user
Pre-conditions	Important action occurs
Post-conditions	Real-time notification is sent
Priority	Can have
Specificity	Sea level (user goal)
Source	Information

Table 55: Use case specification 06.05 - Web notifications

06.06 Read notification	
Primary Actors	Authenticated users
Secondary Actors	-
Description	Users must be able to read received notifications.
Flow	1. Click on the notification 2. The notification redirects the user to the page containing the information relative to the action
Pre-conditions	Received notification.
Post-conditions	Notification is marked as read.
Priority	Can have
Specificity	Sea level (user goal)
Source	Information

Table 56: Use case specification 06.06 - Read notification

07.01 Create new user	
Primary Actors	System Administrators
Secondary Actors	Any user
Description	The platform must enable the creation of new user accounts To note that creation of student accounts is made by external scripts and will not be addressed on this document. The fields here represented are susceptible to modification.
Flow	1. Fill the name 2. Fill the email 3. Fill the user number 4. (Optional) Fill the username 5. (Optional) Fill the internal phone number 6. (Optional) Choose the groups that person belongs to from the list 7. (Optional) Choose the office that person belongs to 8. Click **save changes** button
Pre-conditions	-
Post-conditions	User data saved
Priority	Should have
Specificity	Sea level (user goal)
Source	Information management

Table 57: Use case specification 07.01 - Create new user

07.02 Edit building information	
Primary Actors	Secretary/System Administrators
Secondary Actors	-
Description	The platform must allow creation and deletion of department places and their respective information.
Flow	This can be divided into 2 flows: • Creation 1. Click on the **create building item** button 2. Select the floor 3. Select the tower 4. Insert the item name 5. Choose place type (room, door, stairs, etc) 6. Select the position (click on a point in the floor map) 7. (optional) Choose the person in charge 8. (optional) Insert the telephone 9. (optional) Insert the area 10. (optional) Insert the number of chairs 11. (optional) Insert the number of tables 12. (optional) Choose the Access Point accessible in that place 13. Click the **save place** button • Deletion 1. Click the **remove** button on the place of the list 2. Click the **confirm** button on the box dialog
Pre-conditions	Be on the building information page
Post-conditions	Place changes are updated
Priority	Should have
Specificity	Sea level (user goal)
Source	Information management

Table 58: Use case specification 07.02 - Edit building information

07.03 Edit personal information	
Primary Actors	Any user
Secondary Actors	-
Description	Any user can change their personal information.
Flow	1. Click the **change personal information** button on user's profile 2. (Optional) Change phone number 3. (Optional) Change office 4. (Optional) Change description 5. Click the **update information** button
Pre-conditions	-
Post-conditions	User information is updated.
Priority	Should have
Specificity	Sea level (user goal)
Source	Information management

Table 59: Use case specification 07.03 - Edit personal information

07.04 Upload user photo	
Primary Actors	Any user
Secondary Actors	-
Description	Any user can upload their profile photo. This photo is used to easily identify the user on the platform.
Flow	1. Click the **upload photo** button on the user profile 2. Choose the photo wanted 3. Click the **upload** button
Pre-conditions	-
Post-conditions	User photo is uploaded and updated.
Priority	Should have
Specificity	Sea level (user goal)
Source	Information management

Table 60: Use case specification 07.04 - Upload user photo

07.05 Insert external activity	
Primary Actors	Secretary
Secondary Actors	-
Description	The system must be hable to provide a way to register external activity relative to the DEEC.
Flow	1. Fill **who** was responsible for the activity 2. Fill **where** the activity took place 3. Choose a **category** from the list (Newspaper mention, External divulgation) 4. Choose **when** the activity took action 5. Fill the **scope** of the activity 6. Draw some **comments** about the activity 7. Click **save** button
Pre-conditions	-
Post-conditions	External activity information is saved.
Priority	Should have
Specificity	Sea level (user goal)
Source	Information management

Table 61: Use case specification 07.05 - Insert external activity

07.06 View external activity	
Primary Actors	Secretary
Secondary Actors	-
Description	The system must be able to provide a way to list and view external activity inserted.
Flow	1. Click **view external activity** on the menu bar 2. (Optional) click any activity to see its details
Pre-conditions	-
Post-conditions	-
Priority	Should have
Specificity	Sea level (user goal)
Source	Information management

Table 62: Use case specification 07.06 - View external activity

08.01 Be another user	
Primary Actors	System Administrators
Secondary Actors	Any user
Description	System Administrators must be able to view and act in the platform as another user (excluding another system administrator).
Flow	1. Click on the **be another user** button 2. The administrator is automatically logged out 3. A page of the login is displayed 4. Administrator inserts the email of the other user 5. Administrator clicks on the **login** button 6. Login does not require password and bypasses the user authentication 7. Administrator is browsing the application as another user
Pre-conditions	User is on the system administration page.
Post-conditions	-
Priority	Should have
Specificity	Sea level (user goal)
Source	Utilities

Table 63: Use case specification 08.01 - Be another user

08.02 System logging	
Primary Actors	System Administrators
Secondary Actors	Any user
Description	All important actions in the system must be logged and available to be accessed by the system administrators. This includes the possibility of different levels of logging (configurable by the system administrators).
Flow	This can be divided into 2 different flows: • Change log settings 1. Click the **system logs settings** button 2. Select the logging level 3. Click the **save** button • View logs 1. Click the **system logs** button 2. (Optional) Apply queries to the logs
Pre-conditions	User is on the system administration page.
Post-conditions	Settings are changed.
Priority	Should have
Specificity	Sea level (user goal)
Source	Utilities

Table 64: Use case specification 08.02 - System logging

08.03 Change language	
Primary Actors	Authenticated users
Secondary Actors	-
Description	Users must be able to change the language of the page. They can choose between Portuguese and English. The default will be Portuguese.
Flow	1. Click the language name on the top of the screen
Pre-conditions	-
Post-conditions	Language is changed
Priority	Can have
Specificity	Sea level (user goal)
Source	Utilities

Table 65: Use case specification 08.03 - Change language

09.01 Request key	
Primary Actors	Authenticated users
Secondary Actors	-
Description	All authenticated users must be able to request a key on the platform. Requests are associated with a single door, and can be authorized or denied by key managers.
Flow	1. Select the room from the list 2. Select the start date from the list 3. Select the end date from the list 4. Fill the justification for the request 5. Click the **submit request** button
Pre-conditions	-
Post-conditions	Request is saved and a notification is sent to all key managers.
Priority	Must have
Specificity	Sea level (user goal)
Source	Key request

Table 66: Use case specification 09.01 - Request key

09.02 Change the status of a key request	
Primary Actors	Key managers
Secondary Actors	-
Description	Key managers must be able to change the status to **authorized** or **denied**. Furthermore, when a request is authorized, key managers can change the status of the request to **lent** and finally to **returned**. Note: Once the status is changed to **denied** or **returned**, it cannot be changed anymore.
Flow	1. Select the request 2. Change the status accordingly with the status rules 3. Confirm the action in the confirmation dialogue
Pre-conditions	There is at least one request.
Post-conditions	Request status is changed and the request issuer is notified.
Priority	Must have
Specificity	Sea level (user goal)
Source	Key request

Table 67: Use case specification 09.02 - Change the status of a key request

09.03 Directly lend a key	
Primary Actors	Key managers
Secondary Actors	-
Description	Key managers must be able to directly lend a key without prior authorization.
Flow	1. Select the room from the list 2. Select the user to whom the key will be lent 3. Selected the limit date to return the key 4. Click the **save** button
Pre-conditions	-
Post-conditions	Request is saved.
Priority	Must have
Specificity	Sea level (user goal)
Source	Key request

Table 68: Use case specification 09.03 - Directly lend a key

B Functional tests to the API

This appendix contains test cases for the API testing. Each test case states the use case, API method, its description, the expected outcome and the final result of the test.

Api Endpoint	User				
Test Id	Use case	Method	Description	Expected output	Result
User#001	#01.01	POST /user/login	Login using valid credentials	200: Returns the access token	200: Passed
User#002	#01.01	POST /user/login	Login using incorrect password	401: Error	401: Passed
User#003	#01.01	POST /user/login	Login using empty credentials	400: Error	400: Passed
User#004	#01.02	POST /user/logout	Logout using access_token	200: Empty message	200: Passed
User#005	#01.02	POST /user/logout	Logout using invalid/empty access_token	400: Error	400: Passed
User#006	#07.04	POST /user/uploadPhoto	Upload a valid image	200: Return new photo url	200: Passed
User#007	#07.04	POST /user/uploadPhoto	Upload an invalid image	400: Error	400: Passed
User#008	#07.04	POST /user/uploadPhoto	Upload an oversized image	400: Error	400: Passed
User#009	#06.01	GET /user/[id]	View user profile using a a valid user id	200: User information	200: Passed
User#010	#06.01	GET /user/[id]	View user profile using a a non existing user id	404: Not found	404: Passed
User#011	#07.03	PATCH /users/updateOwnInfo	Update information using valid fields (phoneNumber, biography, office)	200: Information updated	200: Passed
User#012	#08.01	POST /users/beAnotherUser	Send a valid user id	200: Access token to access the user account	200: Passed
User#013	#08.01	POST /users/beAnotherUser	Send an invalid user id	404: Not found	400: Passed
User#014	#06.02	GET /users/-contacts	Retrieve user contacts	200: Returns an array of contacts	200: Passed
User#015	#07.01	POST /users/	Create user with the valid fields	200: User is created	200: Passed
User#016	#07.01	POST /users/	Create user with the invalid fields	422: Validation error	422: Passed

Table 69: Tests for the User model.

Api Endpoint	AuthRule				
Test Id	Use case	Method	Description	Expected output	Result
AuthRule#001	#2.04	POST /authRules/{id}	Create a rule related to a feature passing the role id and feature name as params	200: Rule is added	200: Passed
AuthRule#002	#2.04	POST /authRules/{id}	Create a rule related to a feature passing the role id and a non existent feature name as params	400: Error	400: Passed
AuthRule#003	#2.04	POST /authRules/{id}	Create a rule related to a feature passing a non existent role id and a feature name as params	400: Error	400: Passed
AuthRule#004	#2.04	DELETE /authRules/{id}	Delete a rule related to a role	200: Rule is deleted	200: Passed

Table 70: Tests for the AuthRule model.

Api Endpoint	AuthRole				
Test Id	Use case	Method	Description	Expected output	Result
AuthRole#001	#02.02	POST /authRoles/	Creating a new role passing a valid name and a description	200: Role is added	200: Passed
AuthRole#002	#02.02	POST /authRoles/	Creating a new role passing an already existen name and a description	409: Conflict error	409: Passed
AuthRole#003	#02.03	DELETE /authRoles/{id}	Delete a role passing an id for a non default role	200: Role is deleted and all the rules associated with it	200: Passed
AuthRole#004	#02.03	DELETE /authRoles/{id}	Delete a role passing an id for a default role	400: Error	400: Passed

Table 71: Tests for the AuthRole model.

Api Endpoint	AuthRoleMapping				
Test Id	Use case	Method	Description	Expected output	Result
AuthRoleMapping #001	#02.01	POST /authRoleMappings/	Create a new association between a user and a role	200: Association is added	200: Passed
AuthRoleMapping #002	#02.01	POST /authRoleMappings/	Create a new association between a non existent user and a role	400: Error	400: Passed
AuthRoleMapping #003	#02.01	POST /authRoleMappings/	Create a new association between a existent user and a non existent role	400: Error	400: Passed
AuthRoleMapping #004	#02.01-extra	DELETE /authRoleMappings/{id}	Delete an existing relation between a user and a role using a existent mapping id	200: Association is deleted	200: Passed
AuthRoleMapping #005	#02.01-extra	DELETE /authRoleMappings/{id}	Delete an existing relation between a user and a role using a non existent mapping id	404: Not found error	404: passed

Table 72: Tests for the AuthRoleMapping model.

Api Endpoint	Feature				
Test Id	Use case	Method	Description	Expected output	Result
Feature#001	#02-general	GET /features/allowed/	Get the allowed features being unauthenticated	200: Only the login feature is allowed	200: Passed
Feature#002	#02-general	GET /features/allowed/	Get the allowed features being in a role with all permissions	200: All the features are available	200: Passed
Feature#003	#02-general	GET /features/allowed/	Get the allowed features being in a role that only can perform the "list dissertations" feature	200: Only the "list dissertations" feature is available	200: Passed

Table 73: Tests for the Feature model.

Api Endpoint	AccessCard				
Test Id	Use case	Method	Description	Expected output	Result
AccessCard#001	#04.03	POST /access-cards/	Create a new card with a non existent card number	200: Card is created and returned	200: Passed
AccessCard#002	#04.03	POST /access-cards/	Create a new card with an existent card number	409: Conflict	409: Passed
AccessCard#003	#04.04	DELETE /accesscards/{id}	Delete a card using a valid id	200: card is deleted	200: Passed
AccessCard#004	#04.04	DELETE /accesscards/{id}	Delete a card using an valid id	400: Error	400: Passed
AccessCard#005	#04.03-extra	PATCH /access-cards/{id}	Update card name	200: Success	200: Passed
AccessCard#006	#04.02	GET /access-Cards/{id}	Get card information with valid id	200: Return the card info	200: Passed
AccessCard#007	#04.02	GET /access-Cards/{id}	Get card information with invalid id	404: Card not found	404: Passed

Table 74: Tests for the AccessCard model.

Api Endpoint	AccessRule				
Test Id	Use case	Method	Description	Expected output	Result
AccessRule#001	#04.02	POST /access-rules/	Add a new rule associated with a card and a door	200: Rule is added	200: Passed
AccessRule#002	#04.02	POST /access-rules/	Add a new rule associated with a card using an indexistent door	400: Error	400: Error
AccessRule#003	#04.02	POST /access-rules/	Add a new rule associated with a non existent card using and a door	400: Error	400: Error

Table 75: Tests for the AccessRule model.

Api Endpoint	AccessHolder				
Test Id	Use case	Method	Description	Expected output	Result
AccessHolder#001	#04.06	GET /accessHolders/{id}/filter={include: [accessEvents]}	Get events from an access holder	200: Returns access holder and its events	200: Passed
AccessHolder#002	#04.06	GET /accessHolders/{id}/filter={include: [accessEvents]}	Get events from an invalid access holder	404: AccessHolder not found	404: Passed

Table 76: Tests for the AccessHolder model.

Api Endpoint	AccessAlarm				
Test Id	Use case	Method	Description	Expected output	Result
AccessAlarm#001	#04.05	GET /accessAlarms/ filter={ where: {door: [doorId]}}	Get unauthorized card events for a door	200: Returns the events	200: Passed
AccessAlarm#002	#04.05	GET /accessAlarms/ filter={ where: {door: [doorId]}}	Get unauthorized card events for a non existing door	200: Returns 0 events (does not throw a 404 because we are querying the AccessAlarm object)	200: Passed

Table 77: Tests for the AccessAlarm model.

Api Endpoint	AccessEvent				
Test Id	Use case	Method	Description	Expected output	Result
AccessEvent#001	#04.07	GET /accessEvents/ filter={ where: {door: [doorId]}}	Get events for a door	200: Returns the events	200: Passed
AccessEvent#002	#04.07	GET /accessEvents/ filter={ where: {door: [doorId]}}	Get events for a non existing door	200: Returns 0 events (does not throw a 404 because we are querying the AccessEvent object)	200: Passed

Table 78: Tests for the AccessEvent model.

Api Endpoint	AccessDoor				
Test Id	Use case	Method	Description	Expected output	Result
AccessEvent#001	#04.08	GET /accessDoor/ filter={include: "accessHolders"}	Get all the doors and access holders who have access to it	200: Returns the doors	200: Passed
AccessEvent#002	#04.07	GET /accessEvents/ filter={ where: {door: [doorId]}}	Get events for a non existing door	200: Returns 0 events (does not throw a 404 because we are querying the AccessEvent object)	200: Passed

Table 79: Tests for the AccessDoor model.

Api Endpoint	Dissertation				
Test Id	Use case	Method	Description	Expected output	Result
Dissertation#001	#03.01, #03.15	GET /dissertations/	Get all the dissertations	200: Returns the dissertations	200: Passed
Dissertation#002	#03.01	GET /dissertations/?filter={offset, limit}	Get dissertations by offset and limit	200: Returns the dissertations beginning at offset, the array returned is the size of the limit	200: Passed
Dissertation#003	#03.01	GET /dissertations/?filter={ where: {semester: [semester]}}	Get dissertations for a semester	200: Returns the dissertations for that semester	200: Passed
Dissertation#004	#03.01	GET /dissertations/?filter={ where: {specialization: [specializationId]}}	Get dissertations for a specialization	200: Returns the dissertations for that specialization	200: Passed
Dissertation#005	#03.01	GET /dissertations/? filter={ where: {status: [status]}}	Get dissertations that have a status	200: Returns the dissertations with status = status	200: Passed
Dissertation#006	#03.02	GET /dissertations/[id]	Get dissertation by valid id	200: Returns the dissertation	200: Passed
Dissertation#007	#03.02	GET /dissertations/[id]	Get dissertation by invalid id	404: Not found	404: Passed
Dissertation#008	#03.03	POST /dissertations/	Create a dissertation with all required fields	200: Created	200: Passed
Dissertation#009	#03.03	POST /dissertations/	Create dissertation without some required fields	422: validation error	422: Passed
Dissertation#010	#03.03	POST /dissertations/	Create dissertation passing the id of an eligible student	200. Created and student is associated	200: Passed
Dissertation#011	#03.03	POST /dissertations/	Create dissertation passing the id of non eligible student	400: Error	400: Passed
Dissertation#012	#03.03	POST /dissertations/	Create dissertation passing the id of student associated with another dissertation	400: Error	400: Passed
Dissertation#013	#03.04	DELETE /dissertations/[id]	Delete dissertation by id	200: Dissertation is deleted	200: Passed

Table 80: Tests for the Dissertation model - part I.

Api Endpoint	Dissertation				
Dissertation#014	#03.04	DELETE /dissertations/[id]	Delete dissertation by non existent id	404: Not found	404: Passed
Dissertation#015	#03.03	PATCH /dissertations/[id]	Change dissertation fields	200: Patched	200: Passed
Dissertation#016	#03.03	PATCH /dissertations/[id]	Change the dissertation, passing the id of an eligible student	200: Patched and student is associated	200: Passed
Dissertation#017	#03.03	PATCH /dissertations/[id]	Change the dissertation, passing the id of a non eligible student	400: Error	400: Passed
Dissertation#018	#03.05	PATCH /dissertations/[id]	Change the dissertation visibility to false	200: Dissertation does not appear on dissertations list	200: Passed
Dissertation#019	#03.05	PATCH /dissertations/[id]	Change the dissertation visibility to true	200: Dissertation appears on the dissertations list	200: Passed
Dissertation#020	#03.06	POST /dissertations/	Send a dissertation object to be cloned	200: Created. Dissertation semester and student are empty	200: Passed
Dissertation#021	#03.11	GET /dissertations/?filter={ where: {supervisorId: [supervisorId]}}	List dissertations by supervisor	200: Dissertations from that supervisor are returned	200: Passed
Dissertation#022	#03.11	GET /dissertations/?filter={ where: {studentId: [studentId]}}	Get the dissertation by a student id that is associated with a dissertation	200: Dissertation is returned	200: Passed
Dissertation#023	#03.11	GET /dissertations/?filter={ where: {studentId: [studentId]}}	Get the dissertation by a student id that is associated with a dissertation	200: Empty result	200: Passed

Table 81: Tests for the Dissertation model - part II.

Api Endpoint	ExternalSupervisor				
Test Id	Use case	Method	Description	Expected output	Result
ExternalSupervisor #001	#03.03	POST /externalSupervisors/	Create an external supervisor assigned to a dissertation	200: Created	200: Passed
ExternalSupervisor #002	#03.03	POST /externalSupervisors/	Create an external supervisor not passing a valid dissertation id	400: Not created	400: Passed
ExternalSupervisor #003	#03.03	DELETE /externalSupervisors/[id]	Delete external supervisor by id	200: Deleted	400: Passed
ExternalSupervisor #004	#03.03	DELETE /externalSupervisors/ filter={ where: {dissertationId: [dissertationId]}}	Delete all external supervisors belonging to the dissertation id	200: Deleted	200: Passed

Table 82: Tests for the ExternalSupervisor model.

Api Endpoint	InternalSupervisor				
Test Id	Use case	Method	Description	Expected output	Result
InternalSupervisor #001	#03.03	POST /internalSupervisors/	Create an internal supervisor assigned to a dissertation	200: Created	200: Passed
InternalSupervisor #002	#03.03	POST /internalSupervisors/	Create an internal supervisor not passing a valid dissertation id	400: Not created	400: Passed
InternalSupervisor #003	#03.03	DELETE /internalSupervisors/[id]	Delete internal supervisor by id	200: Deleted	400: Passed
InternalSupervisor #004	#03.03	DELETE /internalSupervisors/ filter={ where: {dissertationId: [dissertationId]}}	Delete all internal supervisors belonging to the dissertation id	200: Deleted	200: Passed

Table 83: Tests for the InternalSupervisor model.

Api Endpoint	DiApplication				
Test Id	Use case	Method	Description	Expected output	Result
DiApplication #001	#03.07	POST /diApplications/	Create an application for a dissertation being an eligible student	200: Created	200: Passed
DiApplication #002	#03.08	POST /diApplications/	Create an application for a dissertation not being an eligible student	400: Error	400: Passed
DiApplication #003	#03.09	POST /diApplications/allowed?id=[id]	Accept the application for a dissertation	200: Student is assigned to the dissertation. The application is deleted	200: Passed
DiApplication #004	#03.08	POST /diApplications/allowed?id=[id]	Accept the application for a dissertation that has already an assigned student	400: Error	400: Passed
DiApplication #005	#03.10	GET /diApplications/? filter={ where: {dissertationId: [id]}}	Return application for a specified dissertation	200: Returns applications	200: Passed

Table 84: Tests for the DiApplication model.

Api Endpoint	Eligible				
Test Id	Use case	Method	Description	Expected output	Result
Eligible#001	#03.09	POST /eligibles/eligibles/updateEligible	A valid csv file with valid fields is uploaded	200: Created	200: Passed
Eligible#002	#03.09	POST /eligibles/eligibles/updateEligible	An invalid file is uploaded	400: Error	400: Passed
Eligible#003	#03.09	POST /eligibles/eligibles/updateEligible	A valid csv file is uploaded with invalid fields	400: Error	400: Passed

Table 85: Tests for the Eligible model.

Api Endpoint	Proposal				
Test Id	Use case	Method	Description	Expected output	Result
Proposal#001	#03.13	POST /proposals/	Create a suggestion with the required fields	200: Created	200: Passed
Proposal#002	#03.13	POST /proposals/	Create a suggestion without required files	422: Validation error	422: Passed
Proposal#003	#03.14	GET /proposals/	Get all the suggestions	200: Suggestions are returned	200: Passed
Proposal#004	#03.14	GET /proposals/[id]	Ge suggestion information by id	200: Suggestion info is returned	200: Passed
Proposal#005	#03.14	GET /proposals/[id]	Get suggestion information with an invalid id	404: Not found	404:Passed

Table 86: Tests for the Proposal model.

Api Endpoint	Specialization				
Test Id	Use case	Method	Description	Expected output	Result
Specialization#001	#03.01, #03.03	GET /specializations/	Get all specializations	200: Specializations are returned	200: Passed

Table 87: Tests for the Specialization model.

Api Endpoint	Place				
Test Id	Use case	Method	Description	Expected output	Result
Place#001	#06.03	GET /places/	Get all the places	200: All places are returned	200: Passed
Place#002	#06.03	GET /places/?filter={ where: {floor:[floor]}}	Get all the places from that floor	200: Only places from that floor are returned	200: Passed
Place#003	#06.03	GET /places/?filter={ where: {name:[name]}}	Get all the places filtered by name	200: Only places that contain that name are returned	200: Passed
Place#004	#06.03	GET /places/[id]	Get a place by its id	200: Place information is returned	200: Passed
Place#005	#06.03	GET /places/[id]	Get a place with an invalid id	404: Not found	404: Passed
Place#006	#07.02	POST /places/	Create a new place with all required fields	200: Created	200: Passed
Place#007	#07.02	POST /places/	Create a new place without some required fields	422: Validation error	404: Passed
Place#008	#07.02	DELETE /places/[id]	Delete place by its id	200: Deleted	200: Passed
Place#009	#07.02	PATCH /places/[id]	Update place fields	200: Information is changed	200: Passed

Table 88: Tests for the Place model.

Api Endpoint	ExternalActivity				
Test Id	Use case	Method	Description	Expected output	Result
ExternalActivity #001	#07.06	GET /externalActivities/	Get all external activities	200: All external activities are returned	200: Passed
ExternalActivity #002	#07.05	POST /externalActivities/	Create a new external activity with all required fields	200: Created	200: Passed
ExternalActivity #003	#07.05	POST /externalActivities/	Create a new external activity with invalid fields	422: Validation error	422: Passed

Table 89: Tests for the ExternalActivity model.

Api Endpoint	Log				
Test Id	Use case	Method	Description	Expected output	Result
Log#001	#08.02	GET /logs/	Get all logs	200: All logs are returned	200: Passed

Table 90: Tests for the Log model.

Api Endpoint	KeyRequest				
Test Id	Use case	Method	Description	Expected output	Result
KeyRequest#001	#09.01, #09.03	POST /keyRequests/	Create a new request with required fields	200: Request is created	200: Passed
KeyRequest#002	#09.01, #09.03	POST /keyRequests/	Create request without required fields	422: Validation error	422: Passed
KeyRequest#003	#09.01, #09.03	POST /keyRequests/	Create request for an invalid place	400: Error	400: passed
KeyRequest#004	#09.02	GET /keyRequests/	List all key requests	200: Key requests are returned	200: Passed
KeyRequest#005	#09.02	PATCH /keyRequests/[id]	Change the status of a key request	200: Status is changed	200: Passed
KeyRequest#006	#09.02	PATCH /keyRequests/[id]	Change the status of a key request with an invalid id	404: Status is changed	404: Passed

Table 91: Tests for the KeyRequest model.