Lean Dashboard

José Pedro Jesus, n.º 44805
Hugo Manuel Jacinto Pinheiro, n.º 44886
Tomás Simão Mendes dos Santos, n.º 45363

Supervisors: João Pereira, e-mail: joao.pereira@inetum.world, Inetum
Filipe Freitas, e-mail: ffreitas@cc.isel.ipl.pt

June 14th 2021

Abstract

The amount of information necessary to develop an application nowadays is a lot to keep track of. Suppose you're a project manager and you want to keep track of the information of your project as well as show it to your team but it is scattered across different platforms.

Data ranging from test results to the current sprints can all be puzzle pieces of a much bigger picture and accessing the various platforms only to view this information can be tiring and complicated.This project's goal is to simplify a team's life, by developing an application that allows a project manager to control and show the team members the necessary information for their project.

This application will make use of a single page application, a web server(Web API), as well as a no-SQL database and an ETL procedure used to extract and transform the information from the various data sources.

Keywords: Information Aggregation, ETL, Project Managing, Project Information, Team Management, Single Page Application, Web API, no-SQL Database

Contents

1 Introduction **5**

1.1 Report Organization 5

2 Functionalities **7**

2.1 Implemented functionalities 7

2.2 Research and planning 8

3 Architecture **9**

3.1 Architecture Principles 9

3.2 Software Architecture 10

3.3 Scheme of the back-end 11

3.4 Data Model 13

3.5 Authization module, Backoffice and Access control 16

4 User Experience Research **17**

4.1 Red routes diagram 17

4.2 Information Architecture 19

4.3 Digital Prototype and Usability Tests 20

5 Client Application **22**

5.1 React vs Angular 22

5.2 Client Architecture 22

5.3 Pages and Components 22

5.4 Client Side Error Handling 23

5.5 Client Side Access Control 23

6 User Interface and Functionalities **24**

6.1 User Interface 24

6.2 Sign In Page 25

6.3 Projects Page 26

6.4 Dashboards Page 26

6.5 Dashboard Page 27

6.6 Settings Page 27

6.7 Projects Settings 28

7 Deployment **29**

8 Conclusion **30**

Chapter 1

Introduction

Nowadays, within a company, it is even more important to have an organized and cooperative team with knowledge of all the steps and goals that need to be worked on for the various projects they are currently participating in. Each member of the team must keep track of high amounts of information and since it is not uncommon that when working on a project a couple of platforms are used to keep track and share work done by the various members, useful information can be scattered on a vast amount of platforms, and sometimes even inside a single platform. The information relative to a project can be obtained from different sources and it is all aggregated in one place, readily available to be displayed to a work team to better guide a certain project's development.

The Lean Dashboard Project was developed to help the company's workers keeping track of all the possible tasks for their projects, gathering all the information needed for the various activities from the many sources that are necessary, presenting it on an easy to read and reactive web application. The project at hands was developed in partnership with Inetum[1] and centred around the development of a responsive web application capable of running on a multitude of devices, ranging from smartphones to desktop computers to large screens such as TV's. This web application displays to a work team of a company all the information regarding various projects being worked on. The information being displayed shows the team what needs to be addressed in the project at hands, such as milestones, bugs, and current errors in the project.

The project required the development of 3 principal components: an ETL procedure, a web API and a client application. This report will explain these 3 components in more depth further on. The first two were developed using the Node.JS programming language while the client application was developed using the React library.

1.1 Report Organization

In this report, we explain the implemented solution to the problem at hands, as well as provide a more in-depth analysis of the architecture and technology choices.

In the Functionalities chapter, we will expose the features that were implemented for the project's application. Additionally, we also showcase a couple of aspects regarding planning and research.

Next up, in the Architecture chapter, we will discuss aspects such as the implementation of the software being developed, decisions made by us and how certain problems were solved.

Furthermore, we dedicate a chapter to the research done in regards to User Experience for our application where we display some of the steps we took in preparing our client application in terms of User Experience.

On the Client Application chapter, we explain the reason for the chosen development platform, how the client application works and display some of the web pages.

In the Deployment chapter, we explain the choices made for the deployment platform and how it was made. Finally, in the Conclusion chapter, we expose the learned lessons and future work for forthcoming versions.

Chapter 2

Functionalities

The functionalities section is destined to list and describe the functionalities that the final application provides the user with.

In this section, information such as what was implemented within the application and the research and planning that was made for the project is present.

The solution is a responsive mobile-first web application that allows the user to consult and aggregate the information of various platforms. Inside a project, a manager is be able to add various dashboards that will then have the desired widgets. The widgets are the structures that hold and show the desired information regarding issues, tests, and sprints from platforms such as Jira[2], Squash[3] and Azure[4]. A manager will then be able to add various users to its project so that those users can consult all the dashboards with the relevant information in each widget.

2.1 Implemented functionalities

These are the features implemented within the final application:

- Retrieving the data from the multiple APIs. We can access three supported APIs to obtain the relevant information, these being Jira, Squash and Azure.
- Authentication of local users using the Authization Module. With the help of the module, we can create and edit users, log in and logout users of local accounts.
- In addition to users, we also support roles for the given users: Superuser, Manager and Collaborator.
- Back-office functionality. With this, managers can do actions such as add new members to a project, remove current project members as well as give certain users roles.
- Creating new projects. Authorized users can add new projects to the platform and manage them.
- Creating various Dashboards inside an existing project. After creating a project, it is also possible to include new dashboards in said project.
- Transforming the retrieved data in a widget format.
- Creating a widget and adding it to an existing dashboard with the use of existing templates, that aid the user by showing them the type of existing widgets.

- Automatically updating widgets with the use of a scheduler. Users can set a time period to refresh existing widgets, making sure the most up to date information is being displayed. The process and technology chosen for the scheduler is explained in the back end portion of this report.
- Setting the credentials for all the platforms. When creating a widget, a user can set up the desired credentials, to choose the data source for a specific widget.
- Front-end client application based on React technology.
- Multi language support for the client application.

2.2 Research and planning

In this section, we display the set of aspects, that despite technically not being functionalities, are essential aspects for the development of this project:

- Red routes diagram. This diagram is part of the Experience research further developed ahead.
- Information architecture[6]. A diagram that displays the existing paths/flows of the client application.
- Digital prototype of the User Experience. The Digital prototype developed allowed us to conduct a series of usability tests. Details about these tests are discussed further ahead in the report.
- Study of the React[7] technology for the development of the client application.

Chapter 3

Architecture

This chapter is dedicated to cover topics like the software's architecture, the data model and the principles under which the application was developed.

It contains information on how the application works and how its various modules and components interact with each other, as well as the structure of the necessary information being stored in the database and the application's authentication and authorization flow.

3.1 Architecture Principles

The software solution in the Lean Dashboard is divided into three main components: the ETL[8], the Lean Dashboard Server and the client application.

The ETL component, which stands for Extract, Transform and Load, is responsible for obtaining the information from the various APIs and then transforming it to the desired format, widgets, and then proceeding to store them in a database. Both the topics of widgets and the storage solution being used are addressed in this report in the Data model.

With the use of the ETL procedure in our application, we bring a couple of advantages to the way we interact with the data being displayed. Since having near-real-time information is enough, with the Extract component we can avoid making high amounts of requests to the various APIs and avoiding overload them.

Additionally, by adapting the extracted information (the Transform component) we can have a format of data that closely resembles the type and aspect of details we are trying to show to the user in each widget (we will also showcase the aspect of a widget in our application). Finally, with the information transformed to the desired format, when can then use a No-SQL database to store it and later use that information to display to the users

The Lean Dashboard server ends up serving as a gatherer of information. Inside this server, we have all of the various projects, created by users, and each one will have its dashboards. Dashboards will be used to group the various widgets, and the user is the one to choose which dashboards will contain which widgets (these being the same widgets that we're created by the Extract, Transform and Load procedure).

Lastly, we have the client application. The client application is responsible for showing the user all of its projects. Users (if they are given these permissions) will be able to consult but also edit projects and dashboards. The main goal will be checking on the widgets inside a certain dashboard since those are the objects that contain the information being retrieved and transformed by the ETL.

3.2 Software Architecture

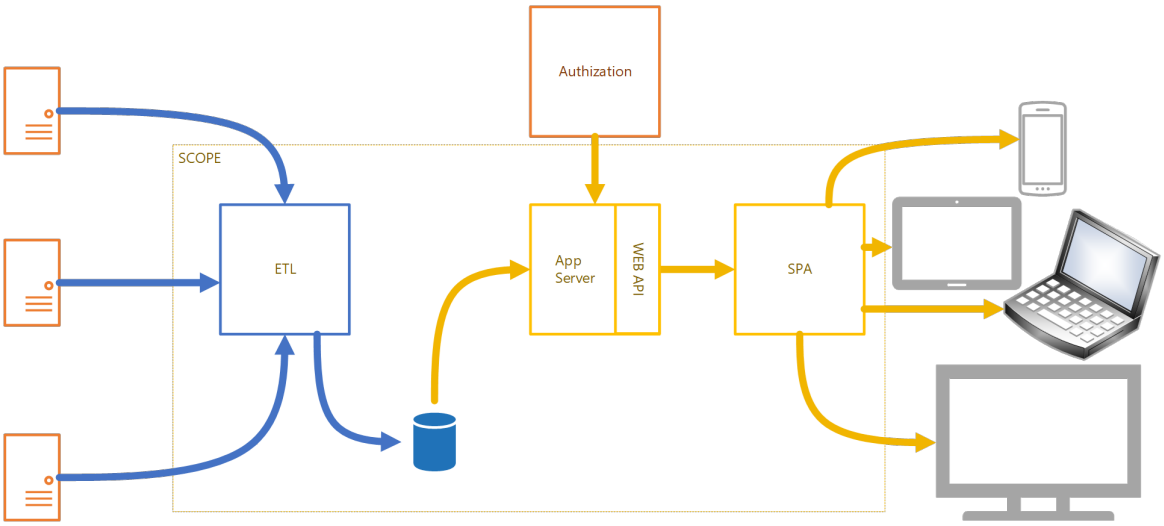

Figure 3.1: Lean Dashboard's Software Architecture and component interactions

On Figure 3.1 it is depicted the various components of our application interacting with each other.

The rightmost part represents the information sources. These sources provide its users with an Application Programming Interface (API), this API will be used by the ETL to obtain solely the needed information.

This information can take different forms. The data objects received from the APIs will differ in terms of structure and information, for example, a project is different from an issue. This means it is crucial to implement an optimized Extract module within the ETL component of our application.

After extracting this information, the ETL will send it towards its respective Transformation module, where it will be treated and converted into a new data object with the necessary fields to represent it visually later.

The Load module of the ETL will grab this newly created Widget object and load it on a database.

The ETL procedure is constantly running and updating the existing widgets on the database, according to the widget's time settings that will be presented in this document.

In the middle of the scheme is the core of the application, the Web Server, and its API.

The Web Server is responsible for processing the requests that are sent by the Single Page Application (SPA), as well as managing the NoSQL database running on ElasticSearch[13]. It interacts with an authorization and authentication module, the Authization module, to manage the users and their roles within the application.

The API supports a varied range of operations that can be called by the user to manipulate the information inside the database. It presents the user with methods to obtain, create, update, and delete the information in the database. The API's endpoints can be found with more depth within Appendix A present in this document.

Lastly on the rightmost side of the diagram, there is the SPA, the component that will be accessed by the application user on their preferred device. To provide the user with the best experience, this SPA was developed as a mobile-first, responsive web application capable of supporting different types of display screens, starting from the smartphone up to the bigger ones like a regular desktop computer or a TV screen.

3.3 Scheme of the back-end

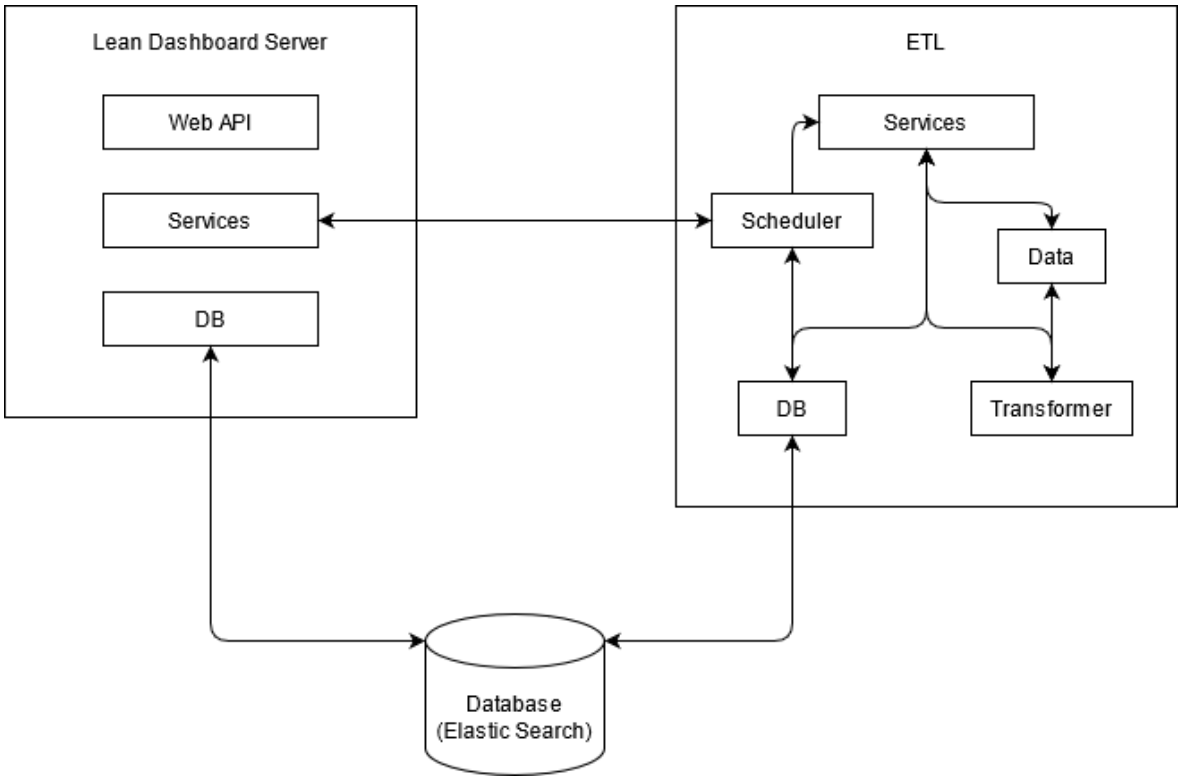

Figure 3.2: Back-end structure and modules interactions

Figure 3.2 depicts the application's back-end and the information flow between its main components. On the left, we have the Lean Dashboard Server and on the right the ETL Procedure, the two key components of the application's back-end.

These two components interact with one another so that the ETL knows what sort of information it should obtain.

The Lean Dashboard Server is composed of 3 Node.JS[14] modules:

- The Web API module is responsible for catching the requests sent to the API, filtering the parameters and body fields, and sending them to the Services module.
- The services module will process every request the Web API receives and forms a function call to the Database(DB) module. If the request consists of modifying or creating a Widget object it will make a call towards the Scheduler module in the ETL.
- The DB module is responsible to access and modify the ElasticSearch based database, containing all the necessary functions to create, obtain, delete and edit the database. It also verifies if the inputted parameters are valid within the context of the application, preventing the existence of duplicates or the association of members to a project who aren't registered in the application, for example.

The ETL component, as we already mentioned, is responsible for accessing the various data sources and transforming them into a specific widget object.

It contains 5 Node.JS modules in its structure:

- The Scheduler module, accessed by the lean-Services module, is responsible for scheduling the automatic ETL process that will update the created widgets according to the time settings a user provides. It accesses the DB module to retrieve the necessary information to execute the correct Services function.

  The module is implemented using the node-cron[15] module so that we can easily create and configure jobs to execute at a set time. It contains 2 Map objects with distinct functionalities, the widgetMap is responsible for providing the scheduler with the necessary ETL-Services function to execute for the various widgets according to the function parameter contained within the widget's structure, and the widgetJobs associates a widgetId to its currently running job so that we can reconfigure them at a later date.

- The ETL-Services module acts as a coordinator for the ETL procedure, it obtains the necessary filtered data from the Data module and then gets the transformed information from the Transformer module.

  After the process of obtaining information is complete it calls the DB module so that this information can be stored.

- The Data module's task is accessing the selected information sources, currently Azure, Jira, and Squash, and sending them to the Transformer module for some light filtering, returning the retrieved object with only the necessary information for its transformation.

- The Transformer module's job is to transform the data into the required information for the widget to be displayed. It transforms the data into widgets such as a Pie Chart, a Data Table, or a Gauge Chart.

- Finally, the DB module is responsible for updating the widget's information with the updated data obtained from the ETL procedure.

3.4 Data Model

For the Data model and the storage of the information, we chose the No-SQL database Elastic Search.

To better facilitate the getting and storage of information, we divided each object into they're own index, as displayed in the following scheme:

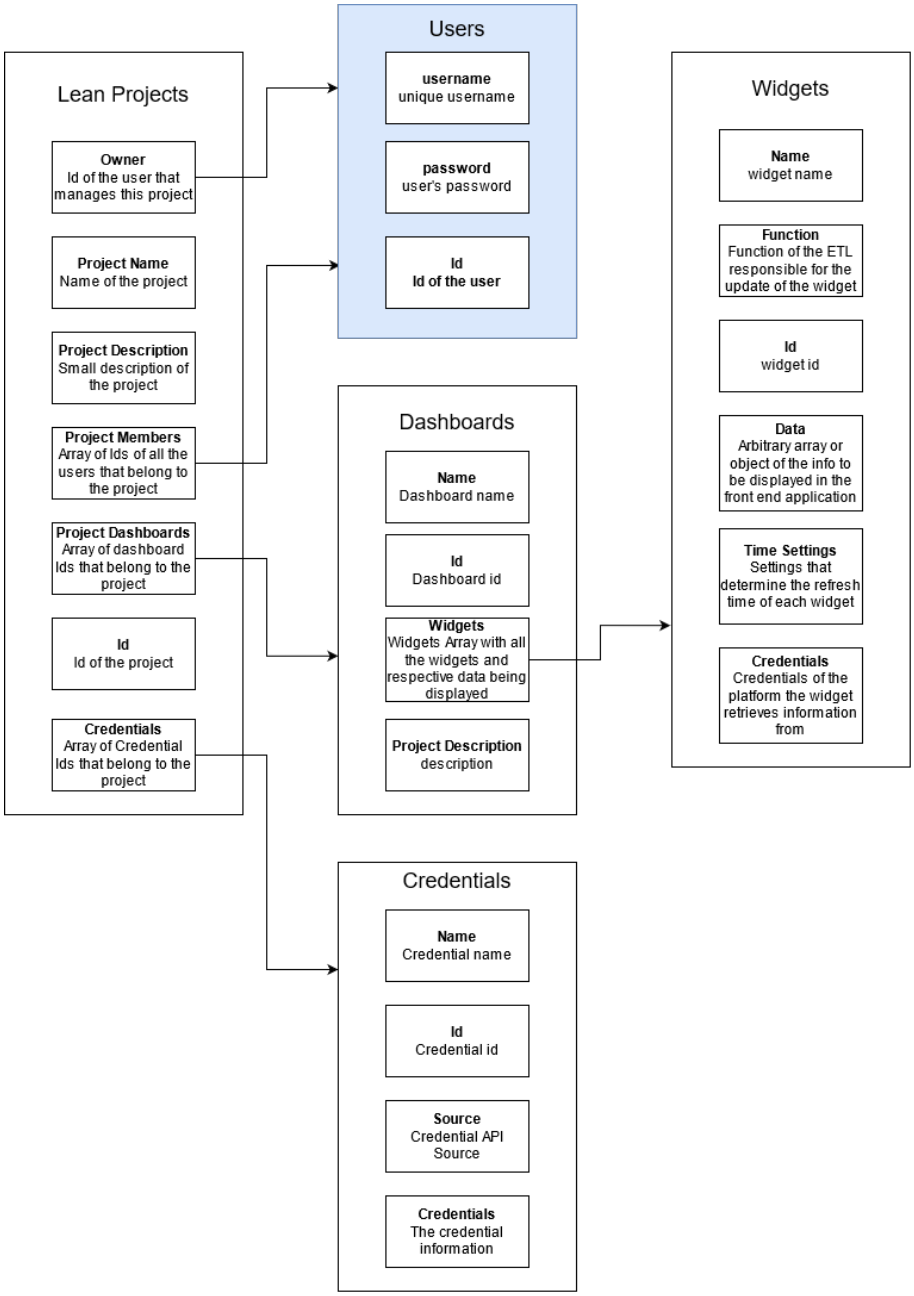

Figure 3.3: Database Data Model

Figure 3.3 depicts the application's data model and the object's structures. Each of these objects contains the necessary fields to store the needed information for the application.

Each object is detailed as the following:

Lean Projects

This is the index where all the Projects created with our application are stored, acting as the main gatherer of information.

It has information over who created it (Owner), the Project's name, its description, and a unique identifier.

Each project can contain a varied number of members, dashboards, and credentials.

This information is stored inside an Array of identifiers, these being Project Members for the members that are currently associated with the project, Project Dashboards for the dashboards, and Credentials for the credentials.

The members simply act as viewers of the information within the project, they can only access dashboards and view the information on the widgets.

Users

The users are highlighted in a different color on the scheme to better differentiate them from the data model since they aren't being stored in the database our application manages and are being stored in the Authization module's database.

Each user will have a username and a password so they can log in to our application, as well as a unique identifier within the database to be used by the array of members within the Lean Projects Objects.

Dashboards

The dashboards are where the widgets will be stored. They don't exist if there isn't at least an already existing Lean Project and will be presented as a web page with all the widgets within the dashboard being displayed.

The dashboard objects contain a name, an Array of Widget identifiers that are associated with the dashboard, a description, and a unique identifier within the database.

Credentials

The credentials consist of a specific object containing a name, a source (Azure, Jira, or Squash), and a credential object containing all the credentials themselves.

This credential object is structured as such for each of the information sources:

Each source requires a different form of credential so that the authentication on their web API is successful. These credentials are mandatory as the ETL procedure cannot access the specified information source's API without them.
According to figure 3.4:

Azure requires an email, a token, and an Instance field related to the project within the API.

Jira requires an email, a token, an APIPath for the project within the API, and an APIVersion that refers to the API version to be used.

Squash only requires a username, a password, and the APIPath.

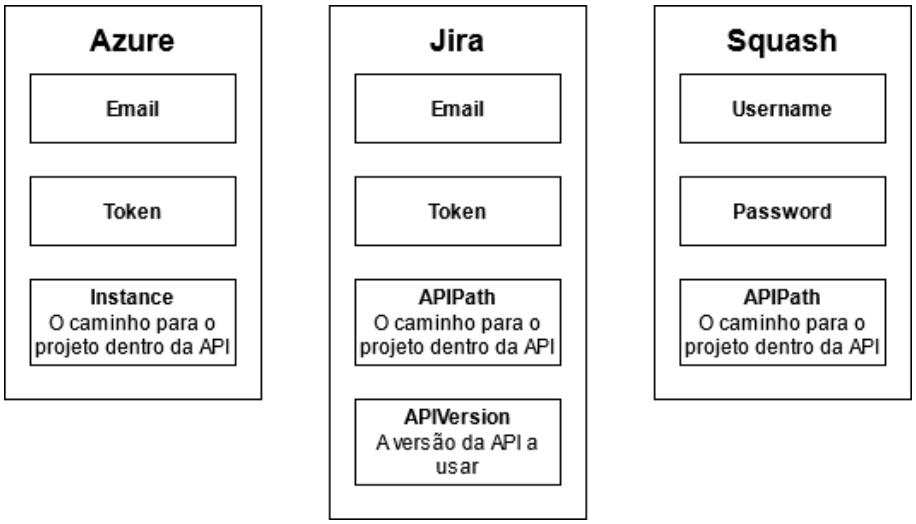

Figure 3.4: Credentials and their structure for each data source

Widgets

The widgets are the result of the ETL procedure.

They are created by the user through the selection of a widget template that will serve as a widget with temporary data to be displayed.

Widgets are made out of several fields like a name, a function, a data object, the time settings for the ETL schedule, the required credentials, and a unique identifier.

The function field refers to which ETL function the scheduler needs to call to start the node job.

The time settings are an object that contains information related to the interval at which the ETL procedure is executed to update the widget.

Each field of this object is structured in CronTab format and consists of:

- seconds;
- minutes;
- hours;
- day of the month;
- month;
- day of the week;

The credentials are the credential object associated with the project.

Without them obtaining information is impossible, making it a required field when creating the widget.

The data object is where all the transformed ETL information will be stored. It is the main reason the application utilizes an ElasticSearch database, as the information contained within this object will vary from widget to widget, making it a hard task to store consistently within a relational database like PostgreSQL.

3.5 Authization module, Backoffice and Access control

To give our users authorization and authentication features we had to think of a way of letting users create accounts, have users with different roles inside the application and have different roles have their privileges. To give users that set of features, and as a request of Inetum, we utilized a Node module called Authization[9].

As a reference point, this module was a project developed last year by the ISEL students Tiago Matias, Diogo Leandro and João Barata of the LEIC programme as a final project of the Project and Seminar course, and was also developed in partnership with Inetum, and serves as a module that allows to set up an RBAC model with various roles, permissions and decide which roles should have access to what permissions. The module will also take care of aspects such as login, logout and the creation of users. Since the module requires a SQL Database for the storage of users, we had to use PostgreSQL[10] to handle the storage of users, the created roles and permissions.

As for the RBAC model, it was requested that there were three types of users: a master user, with every kind of accesses and ultimate control (that includes access to every existing project, both seeing and editing them), a user that would be able to manage certain projects and create new ones and finally a user with only permission to see projects he is a part of. With that said, we came up with an RBAC model with the roles Superuser, Manager and Collaborator, with the following accesses:

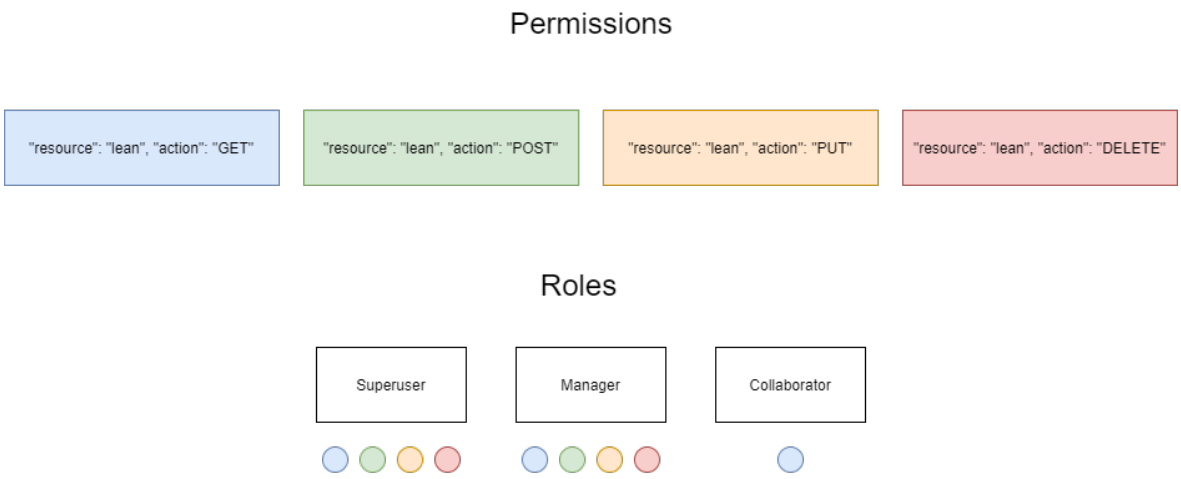

Figure 3.5: Application User Roles and their respective permissions

As said before, we defined three roles. The Superuser will have access to all of the defined permissions, being able to access and modify all projects, without having to be a part of them. Below the Superuser, the Manager role will act in a similar way to the Superuser, but will only be able to modify projects that we're created by himself. Additionally, he will only be able to visualize projects (and the dashboards and widgets associated with the said project) if he is a member or instead, the creator of that project. A manager will be able to manage aspects of a project such as its name, description, containing dashboards and widgets, as well as all the members in it. Finally, the role of Collaborator. Being the role with fewer permissions, users with this role will only be able to access projects they are members of.

Chapter 4

User Experience Research

This section covers our User Experience research and studies on how our application should be controlled.

Our research on every necessary item to develop an application that is easy and intuitive to use will be presented here.

User Experience, commonly called UX, is the term used when referring to how users interact with a certain product. If we use an analogy, if we were to open a door, it is more about how the door handle is used than what shape or colour it has, or what material it is made from.

UX is also greatly impacted by the context a product is being used and the type of users using said product. Inetum conducted a UX research using contextual interviews with potential users of Lean Dashboard in order to understand their real needs. After studying the observations, 2 Personas were created that focus on the needs of the real target. That list of personas would help us take into consideration how we would address the UX design of our early digital prototypes (which later reflect on the client application), by having various types of users with specific personalities, most wanted features and specific positions inside a company.

Following that, we developed a digital prototype that would allow us to make a series of usability tests with real users. Those tests would allow us to better determine what needed to be addressed in our digital prototype and solve issues before any implementation was being done.

With that said, we believe the User Experience research done by the group is something that can greatly improve the result of the client application, whilst saving implementation time by allowing us to make some decisions beforehand. Usually, problems are easier to solve in a digital prototype than they are in code.

4.1 Red routes diagram

A Red Routes matrix is developed to aid a designer to identify the crucial and frequent tasks which users perform with our product. It consists of a matrix that delivers the frequency of performing, as well as the number of users who perform a specific task.

In the process of identifying Red Routes, there are some factors to be considered:

- Critical: Tasks that deliver relevant value to users.
- Frequency: Tasks that are performed at high frequency, usually represent the use cases of over

90% of users.

- Key-value drivers: Red Routes drive the key business metrics.
- Impact: Red Routes affects significantly the overall user experience.

Through identification of our users' top tasks, we can:

- Foresee users' needs.
- Develop a website utilizing users' needs.
- Target fundamental website pages.
- Conduct usability tests.

Regarding the identification of Red Routes' major advantage, it aids the team to identify the most important content and functionality rooted in usefulness to most users. Further, it supports a team in the selection of the minimum viable product (MVP), which leads to a more significant product roadmap design with the purpose of continuous iteration and overcome potential usability barriers on relevant user journeys.

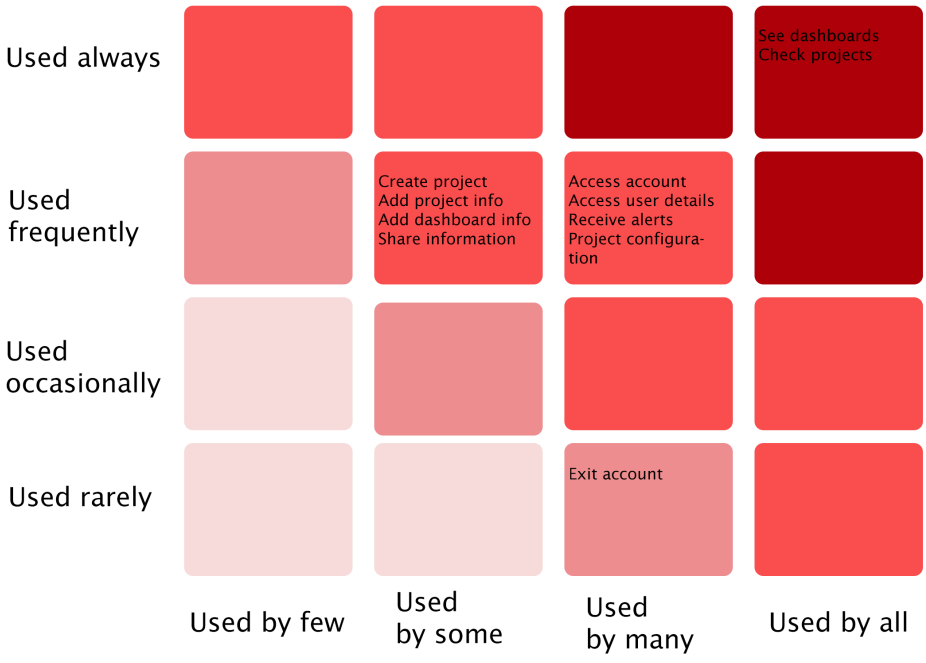

Figure 4.1: Red Routes Diagram

The team designed the Red Routes for Lean Dashboard application, as we can see in figure 4.1, based on the list of Personas that Inetum provided. We took into account the above factors to create the Red Routes in the best way possible.

4.2 Information Architecture

Information architecture(IA) consists of capability of sorting and designing the content of not only the web, but also websites and mobile applications. The main purpose of information architecture is to organize content to facilitate the users' process of tuning to the product's functionality and thus, finding anything they might want easily. There are several elements that can affect the content structure, among which, IA experts tend to regard the specifics of the target audience's needs. This can be justified by the fact that the IA prioritizes user satisfaction. The main factor that leads people to visit different websites is content. It is commonly shared that producing valuable content for users is crucial, but so is ensuring that the content is easily found. We use Personas that Inetum provided as a tool to find the most adequate IA, and ran usability tests.

Figure 4.2: Information Architecture Diagram

With this diagram, we can better plan the making to the various resources by achieving a flowchart that dictates if the various flowchart accesses make sense (and easily correct them if they don't).

4.3 Digital Prototype and Usability Tests

A Digital Prototype is a tool used in UX research to develop a mock user interface that can be utilized in use-case tests. These tests gather a small group of people and establish a task that all users need to complete.

A Digital Prototype is how we validate our basic idea and the premises underpinning it by collecting user feedback. With the obtained results, we can then determine what aspects need to be addressed in the Digital Prototype by us developed.

The usability tests were performed by five Inetum employees, the team used five simple use-cases to test the digital prototype. The use-cases that were used:

- USE CASE 1: A user creates a Lean Dashboard account and creates a project. Then log out of your account.
- USE CASE 2: A user already with an account created, creates a dashboard in an existing project.
- USE CASE 3: A user with an account created creates a widget in an existing project and dashboard.
- USE CASE 4: A user already with an account created, adds a member to his existing project.
- USE CASE 5:A user already with an account created, changes the project name.

After performing all tests, the team changed some web flows, buttons and created new pages based on the interviewee's feedback.

This research and planning done beforehand (before the full implementation of the client application) can greatly decrease implementation costs since it is much easier to make modifications to this mocked user interface than it is to make some of the same changes on the client application code. We utilized the platform Figma[12] to develop our Digital Prototype on figure 4.3:

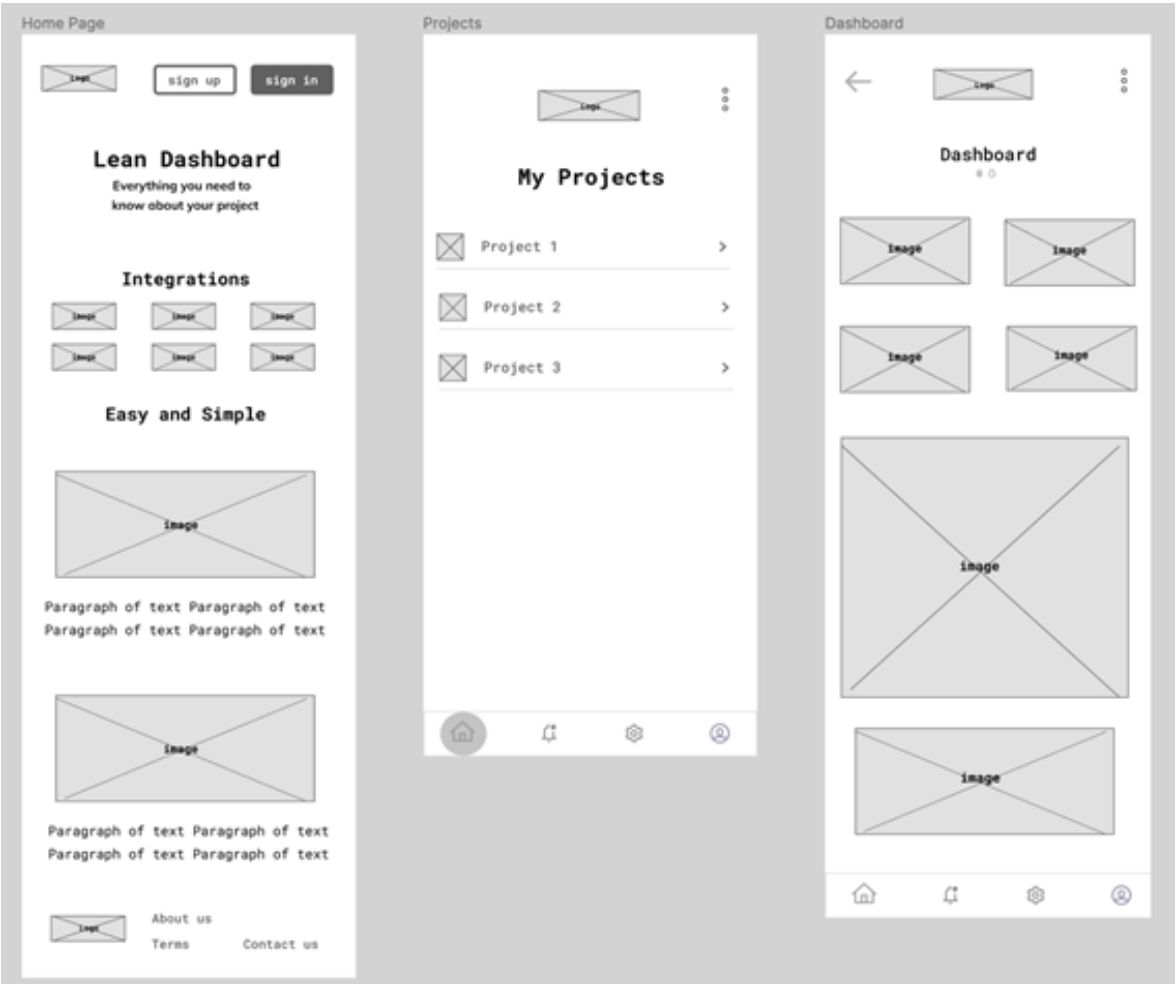

Figure 4.3: Application Digital Prototype

Chapter 5

Client Application

The following section explains the implementation for the client's side of the application.

On this section it is mostly explained the decisions made when implementing the client application, aswell as what each page of the application is used for and how they function.

5.1 React vs Angular

React was our choice over Angular. The reason behind such decision was that Angular has a lot of concepts and syntax to learn, as well as the fact that Angular's documentation is much longer and we believe it might take some time to learn a new language given that the time frame is relatively short. React is easier to learn in the short term than Angular and the group that developed this project would attend a course to learn react, which would be an asset.

React is an open-source JavaScript library focused on the development of user interfaces. With this library developing web applications, SPA (Single Page Application) or mobile applications turns into a very easy task since it offers great benefits in modularity and promotes a very clear flow of data and events. This greatly facilitates the development and planning of complex apps. We make use of the React Router library for the routing of the application. It enables the navigation among views of various components in a React Application and keeps the user interface in sync with the URL. It greatly helps with developing the application's navigation and how each page should look like.

5.2 Client Architecture

The Figure demonstrates how the team organizes the Client-Side code.
The component defines a class that contains data and logic and is associated with an HTML that defines a view.
Pages is a class that connects with Lean Dashboard API to provide data to the components and aggregate the necessary components to be displayed.

5.3 Pages and Components

Components are the building blocks of the React app. The components is a JavaScript function that optionally accepts properties and returns a React element that describes how a section of the

User Interface should appear on the user screen.

Inside the Pages directory, we have the web pages where interact with the server and provide data for the components. This package contains the views of the general pages and uses the components to create these pages.

5.4 Client Side Error Handling

5.5 Client Side Access Control

Chapter 6

User Interface and Functionalities

The following section presents the graphical user interface (UI) of the application.

6.1 User Interface

With help from the information gathered from the user experience tests, the front-end application started development using React.
Based on the low fidelity prototype designed to conduct the tests, we implemented the necessary pages and components for the application.
Each component is designed in a way that makes recycling them on other pages easier and each page was easily and quickly implemented by using the Material UI Framework, an easy-to-use tool that provides pre-made components. Material UI also provides methods and tools to design pages according to the screen size being used to view the client application, making implementing the mobile and desktop pages a much simpler task.
As of now, the client application has multi-language support for 3 languages: English, Portuguese, and French. It uses i18n to translate.

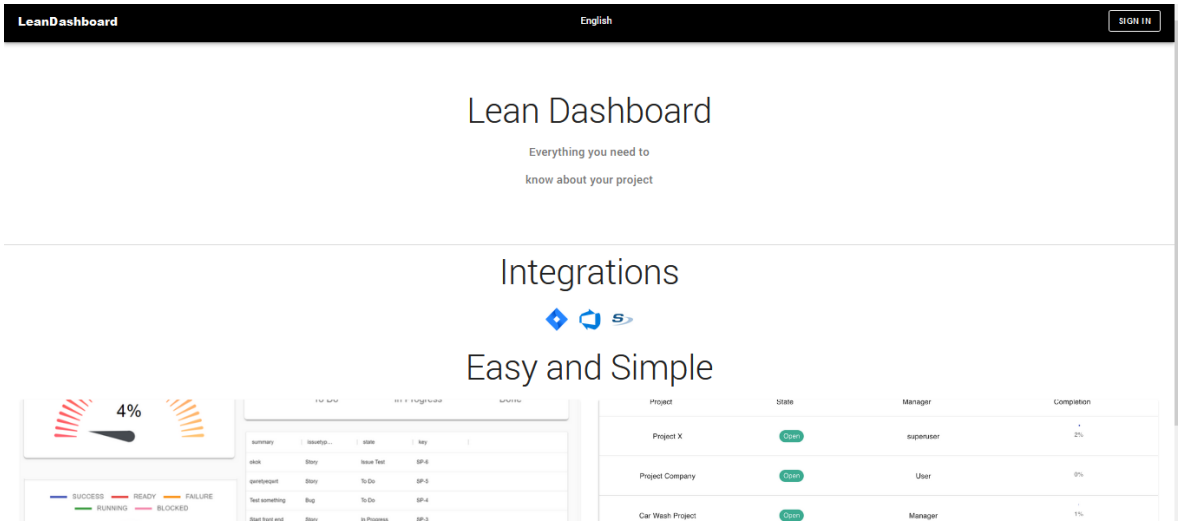

Figure 6.1: Home Page

In figure 6.1, we have the Home Page, it is the first page the user sees. Here the user can change the language of the application and may begin his Log In process, by pressing the Sign-in button.

Once the button is clicked the user will be redirected to the Sign-in page where he will be inserting his credentials to enter the application.

6.2 Sign In Page

The sign in page is where the user enters the credentials necessary to sign into our application. They only have to input a valid username and its associated password to enter.
The page has a checkbox the user can toggle to allow the browser to keep track of the session. The user's session information is stored in two different storages.
The session storage is used when the user does not want the browser to remember them since the information is cleared when they close the tab.
The local storage is used when the user toggles the "Remember Me" checkbox. This storage will save the session information and won't be cleared until the user logs off. So that this effect can be verified, the application verifies if the browser has any saved session, if it's not saved the client redirects the user to the login page if the user is trying to access a restricted page.
The user's username is stored within the application by using the React Hook "UseContext" to later be used for the required API calls. In case the session information is saved, the client sends a request to the API to authenticate the user within the application.

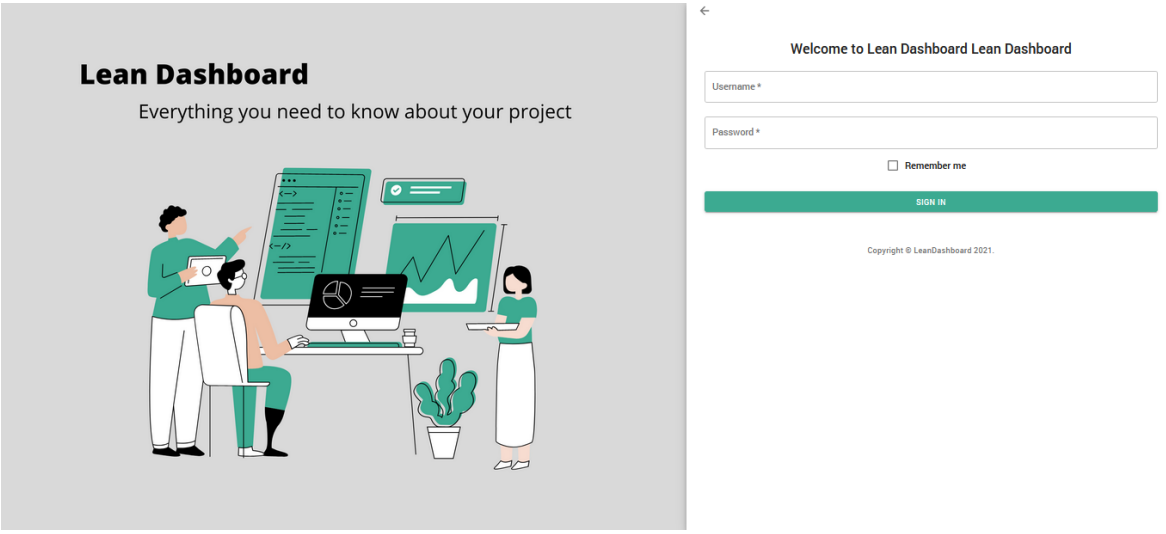

Figure 6.2: Sign In

In figure 6.2, is the view that represents the entrance to the user account. It is composed by a form of two text input boxes that receive the username and the password.

6.3 Projects Page

This view presents to the user the projects that he is on. The page displays in form of a table or list of the projects composed by name, state, manager ,and completion, which is the percentage of completion of the projects. The manager or superuser role can add a project by pressing the button. Once clicked will show up a pop-up to fill with name, description, and end date.

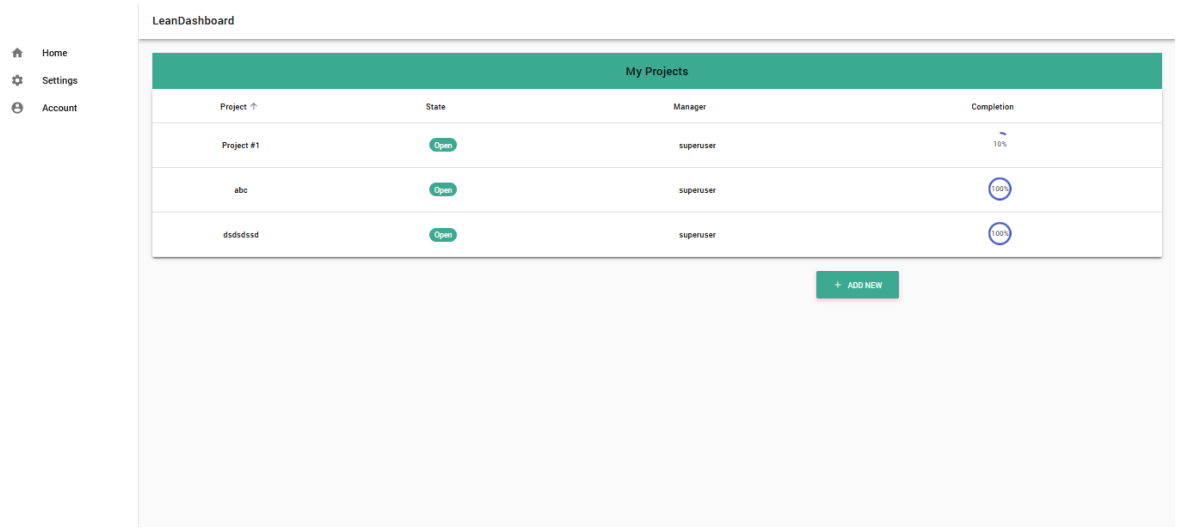

Figure 6.3: Projects Page

6.4 Dashboards Page

This page gives the user a view of the dashboards inside of a project. It can be displayed in form of a list or a group of cards. On this page, we can add a dashboard and be redirected to the project's settings page. On mobile, the way that options appear is with a swipeable drawer and for other screens with a group of buttons displayed on the left side of the screen.

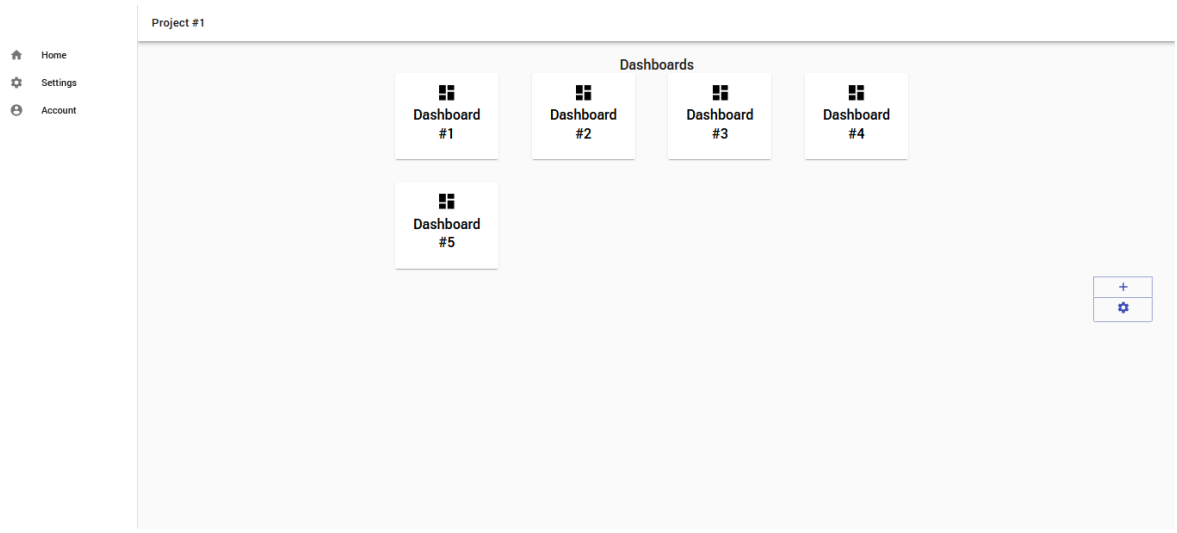

Figure 6.4: Dashboards Page

6.5 Dashboard Page

On this page, the user will see the widgets of one dashboard. This page offers four options in the group of buttons or a swipeable drawer, such as add widgets, dashboard settings, delete a dashboard, and edit widgets. By clicking on the first option, the user will be redirected to a page where all the template widgets that can be added to Lean Dashboard are presented. The second option is a pop-up to change the name and description of a dashboard. The third is a pop-up to delete the dashboard and the fourth button will be redirected to a page similar to the template widgets to edit a specific widget, like deleting it, changing the credentials being used, or the update time.

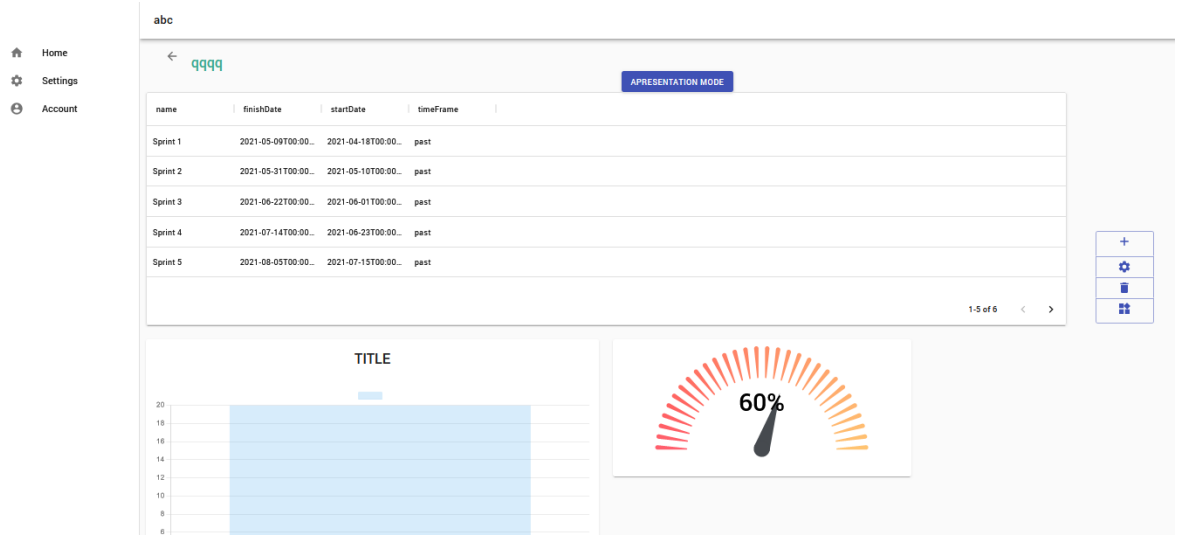

Figure 6.5: Dashboard Page and Widget View

6.6 Settings Page

This view is where the user can change the application's display language. If the user is a superuser, he can add users to the Lean Dashboard application or access the User Management page where all

the users in the application can be viewed and edited, by adding or deleting roles for a specific user, deleting users or changing his username or password.

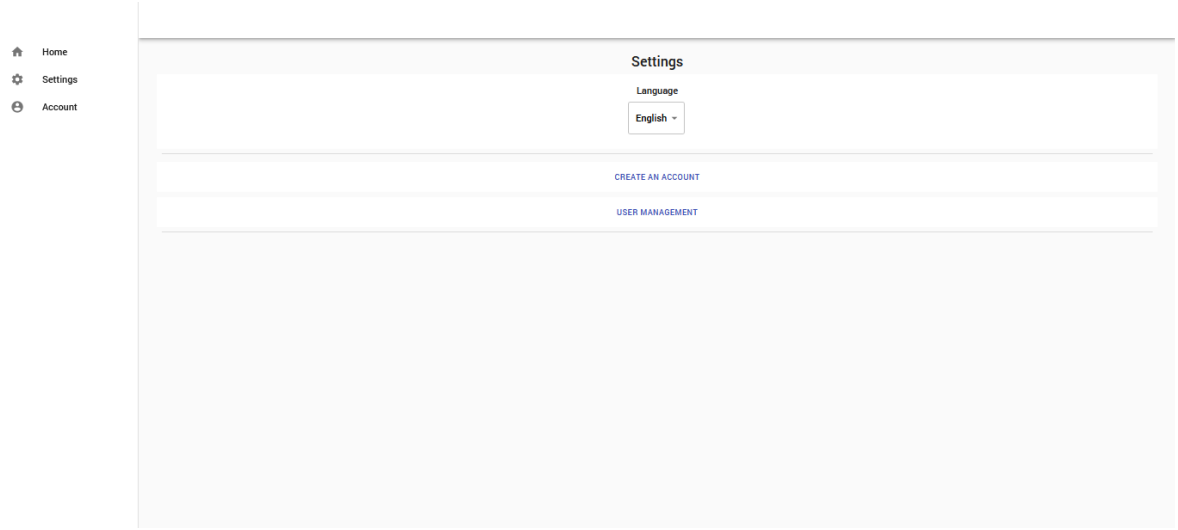

Figure 6.6: Settings Page

6.7 Projects Settings

All the project settings are dealt with on this page, where the project manager or superuser can change the name or description of the projects, add users or platform credentials, that will later be used to add widgets.

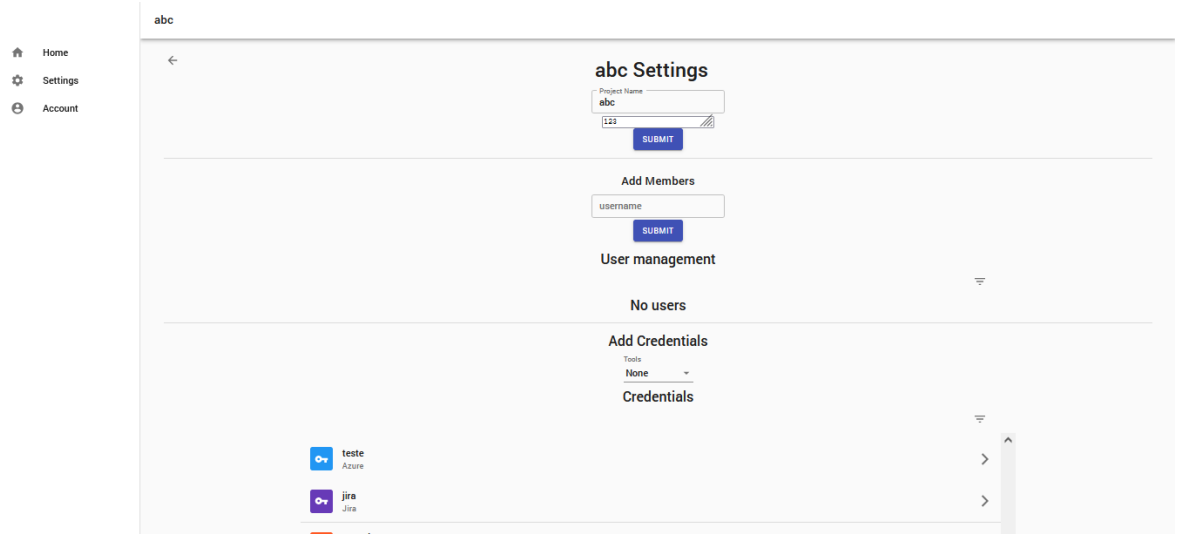

Figure 6.7: Project Settings Page

Chapter 7

Deployment

Chapter 8

Conclusion

Developing this project, allowed the team to apply the experience acquired along the course in this project.

This project provided the opportunity to work with Inetum and developing a web application, from back to front, as well as interacting with other applications in order to learn how to implement the necessary API requests to obtain the ETL data.

The project also taught the group a lot about the research and planning of an application's user experience as the team has never been introduced to such a topic. Conducting the research on UX and usability tests showed us a new perspective on application development.

In the long term, the following versions can have other functionalities like features to the digital accessibility, which means making the application more inclusive for all users, drag and drop the widgets in the dashboard making the dashboard more configurable, and creation of more widgets.

According to our initial planning, some of the tasks took longer than we were expecting, especially the ETL procedure as it took some time to learn and adapt to our application, but overall the application was finished successfully with good results solving the main problem of aggregating all the spread information in one place for a team to see, while having an easy to use and intuitive design that adapts to every type of device and supports various display languages.

Bibliography

[1] Inetum https://gfi.world/pt-en/

[2] Jira- Software development tool used by agile teams https://www.atlassian.com/software/jira

[3] Squash - Suite tool for test management https://www.squashtest.com/?lang=en

[4] Azure - Cloud computing service by Microsoft https://azure.microsoft.com/en-us/

[5] RBAC - Role-based Access Control `https://en.wikipedia.org/wiki/Role-based_access_control`

[6] Information Architecture `https://en.wikipedia.org/wiki/Information_architecture`

[7] React https://reactjs.org/

[8] ETL - Understanding It and Effectively Using It. https://medium.com/hashmapinc/etl-understanding-it-and-effectively-using-it-f827a5b3e54d Consulted on April 1st 2021

[9] Authization Module https://github.com/dleandro/Authentication-and-Authorization-Node-Component-

[10] PostgreSQL: The World's Most Advanced Open Source Relational Database https://www.postgresql.org/

[11] Miro - Platform used for the Information Architecture schema https://miro.com/

[12] Figma Prototype toolhttps://www.figma.com/prototyping/

[13] ElasticSearch https://www.elastic.co/elasticsearch/

[14] Node.JS https://nodejs.org/en/

[15] node-cron https://www.npmjs.com/package/node-cron

Appendix A - API Documentation

Documentation Settings

Lean Dashboard

Lean

Projects

GET /api/lean/projects

http://localhost:8000/api/lean/projects

Get All projects from Lean Dashboard

Example Request /api/lean/projects

```
curl --location --request GET 'http://localhost:8000/api/lean/projects'
```

GET /api/lean/projects/user/{username}

http://localhost:8000/api/lean/projects/user/superuser

Get All projects from Lean Dashboard for a specific user

Example Request /api/lean/projects/user/{username}

```
curl --location --request GET 'http://localhost:8000/api/lean/projects/user/superuser'
```

POST /api/lean/projects

http://localhost:8000/api/lean/projects

Create a project

BODY raw

```
{
    "name": "new project with date from temp user",
    "description": "testing dates for temp user",
    "startDate" : "06-23-2021",
    "endDate" : "06-20-2031"
}
```

Example Request /api/lean/projects

```
curl --location --request POST 'http://localhost:8000/api/lean/projects' \
--data-raw '{
    "name": "new project with date from temp user",
    "description": "testing dates for temp user",
    "startDate" : "06-23-2021",
    "endDate" : "06-20-2031"
}'
```

GET /api/lean/projects/{id}

http://localhost:8000/api/lean/projects/dPfiOXoB7uaFwnIMOTzN

Get a specific project

Example Request /api/lean/projects/{id}

```
curl --location --request GET 'http://localhost:8000/api/lean/projects/dPfiOXoB7uaFwnIMOTzN'
```

PUT /api/lean/projects/{id}

http://localhost:8000/api/lean/projects/U_fZOXoB7uaFwnIMcTwn

Update name or description from a project

BODY raw

```
{
    "name" : "Testing update function",
    "description" : "Testing"
}
```

Example Request /api/lean/projects/{id}

```
curl --location --request PUT 'http://localhost:8000/api/lean/projects/U_fZOXoB7uaFwnIMcTwn' \
--data-raw '{
    "name" : "Testing update function",
    "description" : "Testing"
}'
```

PUT /api/lean/projects/{id}/owner

http://localhost:8000/api/lean/projects/L1Z3TnoBlSjgkE0xGbsI/owner

Update name or description from a project

BODY raw

```
{
    "newOwner": "superuser"
```

Example Request /api/lean/projects/{id}/owner

```
curl --location --request PUT 'http://localhost:8000/api/lean/projects/L1Z3TnoBlSjgkE0xGbsI/owner'
--data-raw '{
    "newOwner": "superuser"
}'
```

DEL /api/lean/projects/{id}

http://localhost:8000/api/lean/projects/O6_3o3kB52sP8UCyIltt

Deletes a project, making it an "Archived" project

Example Request /api/lean/projects/{id}

```
curl --location --request DELETE 'http://localhost:8000/api/lean/projects/O6_3o3kB52sP8UCyIltt'
```

POST /api/lean/projects/:id/users

http://localhost:8000/api/lean/projects/H1ZnTnoBlSjgkE0xobvD/users

BODY raw

```
{
    "username": "temp"
}
```

Example Request /api/lean/projects/:id/users

```
curl --location --request POST 'http://localhost:8000/api/lean/projects/H1ZnTnoBlSjgkE0xobvD/users'
--data-raw '{
    "username": "temp"
}'
```

DEL /api/lean/projects/:id/users/{username}

http://localhost:8000/api/lean/projects/vxuewnkBE0sk1r1Qi0ZN/users/zepedroj

Deletes a specific user from the specific project

Example Request /api/lean/projects/:id/users/{username}

```
curl --location --request DELETE 'http://localhost:8000/api/lean/projects/vxuewnkBE0sk1r1Qi0ZN/user
```

Dashboards

GET /api/lean/projects/{id}/dashboards

http://localhost:8000/api/lean/projects/VpStcnoB7HcdE2OwxI0K/dashboards

Gets all dashboards for a specific project

Example Request /api/lean/projects/{id}/dashboards

```
curl --location --request GET 'http://localhost:8000/api/lean/projects/VpStcnoB7HcdE2OwxI0K/dashboa
```

GET /api/lean/projects/{id}/dashboard/{dashboardId}

http://localhost:8000/api/lean/projects/VpStcnoB7HcdE2OwxI0K/dashboard/V5TNcnoB7HcdE2Ow340a

Gets a specific dashboard for a specific project

Example Request /api/lean/projects/{id}/dashboard/{dashboardId}

```
curl --location --request GET 'http://localhost:8000/api/lean/projects/VpStcnoB7HcdE2OwxI0K/dashboa
```

POST /api/lean/projects/{id}/dashboard

http://localhost:8000/api/lean/projects/ePkBzXkBM-3hOBzM-3-Q/dashboard

Add a dashboard to a Project

BODY raw

```
{
    "name": "New Dashboard",
    "description": "description"
}
```

Example Request /api/lean/projects/{id}/dashboard

```
curl --location --request POST 'http://localhost:8000/api/lean/projects/ePkBzXkBM-3hOBzM-3-Q/dashbo
--data-raw '{
    "name": "New Dashboard",
    "description": "description"
}'
```

DEL /api/lean/projects/{id}/dashboard/{dashboardId}/widgets/{widgetId}

Deletes a specific dashboard from a specific project

Example Request /api/lean/projects/{id}/dashboard/{dashboardId}/widgets/{widgetId}

```
curl --location --request DELETE 'http://localhost:8000/api/lean/projects/ePkBzXkBM-3hOBzM-3-Q/dash
```

PUT /api/lean/projects/{id}/dashboard/{dashboardId}

http://localhost:8000/api/lean/projects/-XDijnkB7Od3lsy2PgdZ/dashboard/-3D6jnkB7Od3lsy2PAfE

Edits the dashboard information

BODY raw

```
{
    "name": "Update Dashboard Lean Dashboard",
    "description": "description Lean Dashboard"
}
```

Example Request /api/lean/projects/{id}/dashboard/{dashboardId}

```
curl --location --request PUT 'http://localhost:8000/api/lean/projects/-XDijnkB7Od3lsy2PgdZ/dashboa
--data-raw '{
    "name": "Update Dashboard Lean Dashboard",
    "description": "description Lean Dashboard"
}'
```

Widgets

GET /api/lean/projects/widgets/templates

http://localhost:8000/api/lean/projects/widgets/templates

Shows all the widget templates for the application

Example Request /api/lean/projects/widgets/templates

```
curl --location --request GET 'http://localhost:8000/api/lean/projects/widgets/templates'
```

POST /api/lean/projects/{id}/dashboard/{dashboardId}/widgets/{widgetIdTemplate}

http://localhost:8000/api/lean/projects/ePkBzXkBM-3hOBzM-3-Q/dashboard/efkCzXkBM-3hOBzMIX-_/widgets/gvkGzXkBM-3hOBzM3n--

Posts a widget in a dashboard

BODY raw

```
{
    "timeSettings": {
        "seconds" : "",
        "minutes" : "*/1",
        "hours" : "*",
        "dayOfMonth" : "*",
        "month" : "*",
        "dayOfWeek" : "*"
    },
    "credentials": "JIRACredentials",
```

View More

Example Request /api/lean/projects/{id}/dashboard/{dashboardId}/widgets/{widgetIdTemplate}

```
curl --location --request POST 'http://localhost:8000/api/lean/projects/eRkBzYkBM_3bOBzM_3_O/dashbo
```

PUT /api/lean/projects/{id}/dashboard/{dashboardId/widgets/{widgetsId}}

http://localhost:8000/api/lean/projects/7GZvv3oBvTSU8GYG9GI3/dashboard/72Zyv3oBvTSU8GYGF2Ib/widgets/-2aHv3oBvTSU8GYGdmKS

Edits a widget's information

BODY raw

```
{
    "timeSettings": {
        "seconds" : "",
        "minutes" : "*/1",
        "hours" : "*",
        "dayOfMonth" : "*",
        "month" : "*",
        "dayOfWeek" : "*"
    },
    "credentials": "teste",
```

View More

Example Request /api/lean/projects/{id}/dashboard/{dashboardId/widgets/{widgetsId}}

```
curl --location --request PUT 'http://localhost:8000/api/lean/projects/7GZvv3oBvTSU8GYG9GI3/dashboa
--data-raw '{
    "timeSettings": {
        "seconds" : "",
        "minutes" : "*/1",
        "hours" : "*",
        "dayOfMonth" : "*",
        "month" : "*",
        "dayOfWeek" : "*"
    },
```

View More

DEL /api/lean/projects/{id}/dashboard/{dashboardId/widgets/{widgetsId}}

http://localhost:8000/api/lean/projects/VpStcnoB7HcdE2OwxI0K/dashboard/V5TNcnoB7HcdE2Ow340a/widgets/W5TPcnoB7HcdE2Owe42S

BODY raw

```
{
    "timeSettings": {
        "seconds" : "",
        "minutes" : "*/1",
        "hours" : "*",
        "dayOfMonth" : "*",
        "month" : "*",
        "dayOfWeek" : "*"
    },
    "credentials": "JIRACredentials"
```

View More

Example Request /api/lean/projects/{id}/dashboard/{dashboardId/widgets/{widgetsId}}

```
curl --location --request DELETE 'http://localhost:8000/api/lean/projects/VpStcnoB7HcdE2OwxI0K/dash
--data-raw '{
    "timeSettings": {
        "seconds" : "",
        "minutes" : "*/1",
        "hours" : "*",
        "dayOfMonth" : "*",
        "month" : "*",
        "dayOfWeek" : "*"
    },
```

View More

GET /api/lean/projects/{projectId}/dashboard/{dashboardId}/widgets/{widgetId}

http://localhost:8000/api/lean/projects/VpStcnoB7HcdE2OwxI0K/dashboard/V5TNcnoB7HcdE2Ow340a/widgets/W5TPcnoB7HcdE2Owe42S

Gets a specific widget

Example Request /api/lean/projects/{projectId}/dashboard/{dashboardId}/widgets/{widgetId}

```
curl --location --request GET 'http://localhost:8000/api/lean/projects/VpStcnoB7HcdE2OwxI0K/dashboa
```

Only the superuser can access these endpoints

GET /api/lean/projects/users

http://localhost:8000/api/lean/users

Gets all users

Example Request /api/lean/projects/users

```
curl --location --request GET 'http://localhost:8000/api/lean/users'
```

GET /api/lean/projects/user/{id}

http://localhost:8000/api/lean/users/1

Gets a specific user by id

Example Request /api/lean/projects/user/{id}

```
curl --location --request GET 'http://localhost:8000/api/lean/users/1'
```

GET /api/lean/projects/user/{username}

http://localhost:8000/api/lean/users/username/zepedros

Get a specific user by username

Example Request /api/lean/projects/user/{username}

```
curl --location --request GET 'http://localhost:8000/api/lean/users/username/zepedros'
```

DEL /api/lean/users/{username}

http://localhost:8000/api/lean/users/temp

Deletes a user

Example Request /api/lean/users/{username}

```
curl --location --request DELETE 'http://localhost:8000/api/lean/users/temp'
```

GET /api/lean/users/{username}/roles

http://localhost:8000/api/lean/users/superuser/roles

Get the user roles

Example Request /api/lean/users/{username}/roles

```
curl --location --request GET 'http://localhost:8000/api/lean/users/superuser/roles'
```

POST /api/lean/users/{username}/roles

http://localhost:8000/api/lean/users/temp/roles

Gives the user a role

BODY raw

```
{
    "role":"manager"
}
```

Example Request /api/lean/users/{username}/roles

```
curl --location --request POST 'http://localhost:8000/api/lean/users/temp/roles' \
--data-raw '{
    "role":"manager"
}'
```

DEL /api/lean/users/{username}/roles/{role}

http://localhost:8000/api/lean/users/zepedros/roles/manager

Deletes a role from the user

BODY raw

```
{
    "role":"manager"
}
```

Example Request /api/lean/users/{username}/roles/{role}

```
curl --location --request DELETE 'http://localhost:8000/api/lean/users/zepedros/roles/manager' \
--data-raw '{
    "role":"manager"
}'
```

POST /api/lean/register

Creates a user

BODY raw

```
{
    "username" : "temp",
    "password"  : "Testpassword123"
}
```

Example Request /api/lean/register

```
curl --location --request POST 'http://localhost:8000/lean/register' \
--data-raw '{
    "username" : "temp",
    "password"  : "Testpassword123"
}'
```

POST /api/lean/login

http://localhost:8000/lean/login

Logs a user onto our application, can be accessed by anyone

BODY raw

```
{
    "username" : "superuser",
    "password"  : "Superuser123"
}
```

Example Request /api/lean/login

```
curl --location --request POST 'http://localhost:8000/lean/login' \
--data-raw '{
    "username" : "superuser",
    "password"  : "Superuser123"
```

POST /api/lean/logout

http://localhost:8000/lean/logout

Logs a user out of the application, can be used by anyone

Example Request /api/lean/logout

```
curl --location --request POST 'http://localhost:8000/lean/logout'
```

PUT /lean/users/{username}/username

http://localhost:8000/lean/users/zepedros/username

Edits an existing user's username

BODY raw

```
{
    "newUsername": "chris"
}
```

Example Request /lean/users/{username}/username

```
curl --location --request PUT 'http://localhost:8000/lean/users/zepedros/username' \
--data-raw '{
    "newUsername": "chris"
}'
```

PUT /lean/users/{username}/password

Edits an existing user's password

BODY raw

```
{
    "newPassword": "Testpassword123"
}
```

Example Request /lean/users/{username}/password

```
curl --location --request PUT 'http://localhost:8000/lean/users/zepedros/password' \
--data-raw '{
    "newPassword": "Testpassword123"
}'
```

Credentials

GET /api/lean/projects/{id}/credentials

http://localhost:8000/api/lean/projects/ePkBzXkBM-3hOBzM-3-Q/credentials

Gets all project credentials

Example Request /api/lean/projects/{id}/credentials

```
curl --location --request GET 'http://localhost:8000/api/lean/projects/ePkBzXkBM-3hOBzM-3-Q/credent
```

GET /api/lean/projects/{id}/credentials/{credentialId}

http://localhost:8000/api/lean/projects/srQlw3kBDPtlKyKYIIgf/credentials/ulQLw3kBDPtlKyKYW4o7

Gets a specific credential from the project

Example Request /api/lean/projects/{id}/credentials/{credentialId}

```
curl --location --request GET 'http://localhost:8000/api/lean/projects/srQIw3kBDPtJKyKYIIqf/credent
```

POST /api/lean/projects/{id}/credentials

http://localhost:8000/api/lean/projects/ePkBzXkBM-3hOBzM-3-Q/credentials

Creates a credential within the project

BODY raw

```
{
    "name" : "abc",
    "source" : "Jira",
    "credential": {
        "email" : "leandashboardproject@gmail.com",
        "token" : "LPcyGdZolN906MvzdwPHF045",
        "APIPath" : "leandashboard.atlassian.net",
        "APIVersion" : 3
    }
}
```

View More

Example Request /api/lean/projects/{id}/credentials

```
curl --location --request POST 'http://localhost:8000/api/lean/projects/ePkBzXkBM-3hOBzM-3-Q/creden
--data-raw '{
    "name" : "abc",
    "source" : "Jira",
    "credential": {
        "email" : "leandashboardproject@gmail.com",
        "token" : "LPcyGdZolN906MvzdwPHF045",
        "APIPath" : "leandashboard.atlassian.net",
        "APIVersion" : 3
    }
```

View More

DEL /api/lean/projects/{id}/credentials/{credentialId}

http://localhost:8000/api/lean/projects/srQIw3kBDPtJKyKYIIqf/credentials/uLQLw3kBDPtJKyKYW4o7

Deletes a credential from the project

Example Request /api/lean/projects/{id}/credentials/{credentialId}

```
curl --location --request DELETE 'http://localhost:8000/api/lean/projects/srQIw3kBDPtJKyKYIIqf/cred
```

PUT /api/lean/projects/{id}/credentials/{credentialId}

http://localhost:8000/api/lean/projects/hkqC0nkBD1XHrm2KaUW7/credentials/iEqD0nkBD1XHrm2KKkVe

Edits a specific credential

BODY raw