# Logo Description automatically generated

**Lean Dashboard**

**Project and seminar Progress Report**

**Computer Engineering bachelor’s degree**

**Authors**

José Pedro Jesus, 44805 , e-mail: zepedro4259@gmail.com, tel.: 963 730 812

Hugo Manuel Pinheiro, 44886 , e-mail: hugomjp28@gmail.com, tel.: 938 973 719

Tomás Mendes dos Santos, 45363 , e-mail: tomassms@gmail.com, tel.: 911 029 224

**Coordinators**

João Pereira, e-mail: joao.pereira@inetum.world, Inetum

Filipe Freitas, e-mail: [ffreitas@cc.isel.ipl.pt](mailto:ffreitas@cc.isel.ipl.pt)

24/05/2021

**Contents**

[1 Introduction 2](#_Toc72744067)

[1.1 - Background 2](#_Toc72744068)

[1.2 - Relevancy 2](#_Toc72744069)

[1.3 – Report Organization 2](#_Toc72744070)

[2 Solution 3](#_Toc72744071)

[2.1 – Functionalities 3](#_Toc72744072)

[2.1.1 – Implemented functionalities 3](#_Toc72744073)

[2.1.2 – Functionalities missing 3](#_Toc72744074)

[2.2 – Architecture 4](#_Toc72744075)

[2.2.1 – Hardware Architecture 4](#_Toc72744076)

[2.2.2 – Software Architecture 4](#_Toc72744077)

[2.2.3 – Data Model 6](#_Toc72744078)

[2.3 - Front-end 10](#_Toc72744079)

# 1 Introduction

## 1.1 - Background

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, making it an easy task to overlook or forget issues that might bring problems to a project later.

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.

## 1.2 - Relevancy

The Lean Dashboard Project will be 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 will centre 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 will display to a work team of a company all the information regarding various projects being worked on.

The information being displayed will show the team what needs to be addressed in the project at hands, such as milestones, bugs, and current errors in the project.

## 1.3 – Report Organization

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

# 2 Solution

## 2.1 – Functionalities

The solution will be a responsive mobile-first web application that will allow to consult and aggregate the information of various platforms. Inside a project, a manager will 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, Squash and Azure. 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.

### 2.1.1 – Implemented functionalities

As of right now, these are the functionalities that are implemented:

* Retrieving the data from the multiple APIs
* Transforming the retrieved data in a widget format
* Creating an isolated widget (without it being added to a dashboard) with the transformed data
* Creating new projects
* Creating various Dashboards inside an existing project
* Partial authentication (creation of a local user, login, and logout)
* Low fidelity model of the front end
* Red routes diagram
* Website map
* Lean Dashboard API partially implemented

### 2.1.2 – Functionalities missing

What needs to be done:

* The management of projects by the back-office manager
* Full authentication and authorization
* Setting the RBAC roles and permission
* Setting the credentials for all the platforms
* Revisiting the structure of widgets
* Client application

## 2.2 – Architecture

### 2.2.1 – Hardware Architecture

Diagram

Description automatically generated

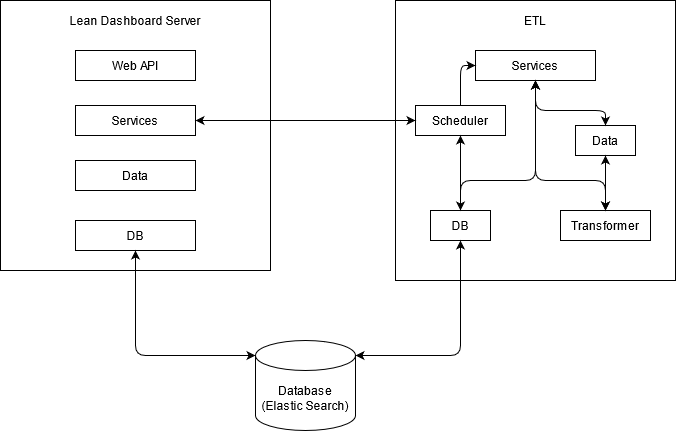
The system being developed is composed by the following modules: the ETL, responsible for obtaining and transforming the information from the various sources of information, the Lean Dashboard API, which will be used to gather all the information regarding the various platforms (such as Jira, Squash and Azure) in the form of widgets, and the application itself to be displayed in the various devices.

### 2.2.2 – Software Architecture

Architecture principles:

* ETL component is responsible for obtaining the information from the various sources and adapting it to widgets. It is a separate model from the application itself
* Lean Dashboard Server serves as a gatherer of information. Inside a Lean Dashboard project, a user can then create various Dashboards that inside have a variety of Widgets (that were created thanks to the ETL)

The back end was designed as following:



Lean Dashboard Server:

* Designed as a Web API for the client to call it when it is developed.
* Accesses the Database to create and manage projects and dashboards.
* Accesses the ETL to create widgets with information and the scheduler to keep these updated.
* …(more bullet points can be added)

ETL:

* Stands for Extract, Transform and Load
* Relevant info is extracted from all the APIs (right now we are using the Jira, Squash and Azure APIs)
* A transformation is applied, in order to store the data in a useful format
* The transformed data is then loaded/stored in a database
* The main module Services gathers all the needed information and executes the needed functions to update and create widgets.
* Data module responsible for the access to the various data sources
* Transformer module responsible for the transformation of the data that it receives into widgets or smaller objects containing the needed information.
* …(more bullet points can be added)

Scheduler:

* The scheduler is an important part of the application
* Implemented using node-cron, offering a great variety of time configurations
* Responsible for the creation of jobs that will run in parallel to the application
* Each widget has a scheduled job associated to it
* The widget’s job is created once the widget is also created
* Every job is executed according to the time settings provided within a widget, the frequency is defined during widget creation or editing
* Contains a map object that maps a function’s name to its call
* The scheduler obtains the function name from the widget information and calls the respective “etl-services” function according to the map object, needed parameters for the function’s call are passed on the array “params” present within a widget’s structure
* The “lean-services” module has another map object that maps the widget’s Identifier to the respective job created by the scheduler so it can be later rescheduled or stopped completely
* …(more bullet points can be added)

### 2.2.3 – Data Model

The application requires a data model and a database to store all the required data.

The chosen database technology was Elastic Search since it provides a flexible no-SQL database. Since in this application there will be a great variety of widgets, all with different structures and data being stored, a NoSQL database such as a Elastic Search came with the advantage of allowing us to store said widgets all in the same index, even with said widgets being different from each other.

A picture containing text

Description automatically generatedThe data model would be organized by the following resources:

#### Lean Projects

* Project objects have their own Id in the Elastic Search index.
* The property “Project Dashboard” represents an array of dashboards
* Only the owner (a manager) can edit the project
* The members are only there as viewers. They can access dashboards and view relevant information
* The main gatherer of information

#### Dashboards

* Dashboards are only present inside the array of dashboards in a project
* They serve as the collection of widgets
* A dashboard would be an entire page being displayed in the client (showing all the widgets present)
* The widgets array will contain the Id of the widgets inside said dashboard

#### Widgets

* Widgets are stored inside an elastic search index and each one has an unique Id
* Widgets are the main information being displayed
* They are the final result of the ETL Procedure
* Besides a few key points, the layout of a widget is arbitrary, because of the different types of data being displayed
* Will contain the necessary information to be displayed in the Lean Dashboard application, such as pie charts, data tables, gauge charts and graphs

#### Widgets Structure

Widgets can be a very abstract concept and somewhat difficult to grasp. Generally, we would go to the various APIs being used (we use the Jira, Squash and Azure ones right now), retrieve the desired information and present it in an application.

The Lean Dashboard application differs from some others in that aspect. The information being stored is a transformed version from the one being retrieved.

For example, I may retrieve from the Squash API all of the tests and their information (such as the names and their status) but if I want to make a pie chart displaying the percentage of tests in each state, I need to adapt/transform the received information in order to obtain the percentage of each test state. That is the transforming process of the ETL procedure. For example, looking at this widget example:

"\_index": "etl-widgets",

                "\_type": "\_doc",

                "\_id": "KAzvnnkBL0mWCmbN6Q2e",

                "\_score": 1.0,

                "\_source": {

                    "name": "Jira sprint gauge chart",

                    "function": "postJiraSprintDateGaugeChart",

                    "source": "Jira",

                    "params": [],

                    "updateTime": {

                        "seconds": "",

                        "minutes": "\*/1",

                        "hours": "\*",

                        "dayOfMonth": "\*",

                        "month": "\*",

                        "dayOfWeek": "\*"

                    },

                    "credentials": "leandashboardproject@gmail.com:LPcyGdZolN906MvzdwPHF045",

                    "data": [

                        {

                            "sprintName": "SP Sprint 2",

                            "info": {

                                "difference\_in\_days": 28,

                                "percentage": 67,

                                "remaining\_days": 9,

                                "past\_days": 19

                            }

                        },

                        {

                            "sprintName": "PROJ Sprint 1",

                            "info": {

                                "difference\_in\_days": 28,

                                "percentage": 67,

                                "remaining\_days": 9,

                                "past\_days": 19

                            }

                        }

                    ]

                }

Looking at that widget, we store a status and the percentage of tests in said status. However, that info can only be obtained after we obtained all of the existing tests, get all the different status and calculate the respective percentage of tests.

## 2.3 - Front-end

One of the blocks of the software architecture is the Client block, which is responsible for the layouts and views that are shown in the Web Application.

At this stage of the project, we have not yet started to implement this block, however, we have already started analysing it. Inetum aided us by establishing a contact with a person with UI experience to help us define the best way to implement the Client block.

The first step in this block was to establish the Red Routes. Red Routes are a matrix which helps prioritise our content and functionality based on usefulness to the majority of users.

Red routes usability:

* Improve speed and effectiveness:
  + Eradicate any usability obstacles on the key users’ journeys
* Describe frequent and critical activities:
  + Important to consider both the frequency and critical nature of the activity

By identifying what the top tasks of our users are, it allows to:

* Anticipate users’ needs
* Guide usability testing
* Target essential websites pages
* Design website with users’ needs in mind

Bubble chart

Description automatically generated with medium confidence

After drawing the red routes, we designed a mind map to organize the flows of our web application. This way, it will be easier to assess if the flowchart makes sense and to change any flow that does not.

Diagram

Description automatically generated

Following the red routes and the mind map, we started to design our web application. In this part, we needed to draw a low fidelity prototype to evaluate whether our flows make sense and if users of our app can easily access them, through the use of usability tests. The most common method in this type of prototype, is to make it on paper and test it with users personally. However, due to the constraints of COVID-19, we decided to design on Figma and make it available online to test with users virtually. At this moment, we are finished designing the low prototype fidelity for testing.

Diagram

Description automatically generated

Further, Inetum gave us user stories that are short, simple descriptions of a feature told from the perspective of the person who desires the new capability, usually a user or customer of the app. With the interpretation of the user stories they gave us, we designed the red routes and flows, similarly to the low fidelity prototype. We will be considering the user stories throughout the remaining of front end design.