Project Report Analysis

UpSkill Java Capgemini

SPRINT 3

BY DUCK SQUAD
DUCK SOFT WORKS & CO.

Daniel Machado
Daniel Lima
Inês Clavel
Thales Lemos

Index

Contents

Project Requirements Description

The system must implement functionalities that allow its users to categorize a given text, obtain information about the language in which that text was written, about the status of the language identification, as well as obtaining information about the tasks in progress and already carried out. This last information can be aggregated, or not, by text category.

The text to be validated must be stored in a text file and must be identified by the URL that allows locating and by one of the existing categories in the information system.

The system must consult a blacklist of web addresses to implement procedures for security concerns about possible malicious file submissions.

The task of identifying a text must be classified in one of the following states:

• Completed: task that resulted in the identification of a language for the provided text.

• Canceled: task that was expressly canceled, or whose processing was not completed in the

time (minutes) stipulated by the user who created it.

• Processing: Task that has been started but has not yet been completed or canceled.

The text categories correspond to areas of interest (e.g., economics, philosophy, mechanics,

nutrition, sport).

The system must identify the user's permissions on the system to make the suitable features.

The system must consist of:

• A backend Web API application that allows managing system information (users,

blacklist of addresses, categories, language identification tasks and their result).

• Text language identification services.

• Web application that works as the frontend of the system and that allows registered users to use the features provided by it.

The frontend application must allow the authentication of a user and, depending on their permissions in the system, to enable the creation, management and consultation of the appropriate business concepts.

Design choices

Backend Architectural Design:

First and for all, the non-functional requirements for this project were to create a Java application with DDD and SpringBoot and it was suggested to use Lucene's Java Library and Aspell Spell Checker to make use of its dictionaries.

For that and to serve the purposes of organization and scalability, we have opted for an **Onion Layered Architecture** in our design choice.

We are also applying the following design patterns:

SOLID design was also taken into consideration by attempting to uphold the "S" - single-responsibility principle and the "O" – open-closed principle.

RDD(Responsibility Driven Design) was applied in conjunction with **GRASP**, **MVC** and **DDD** patterns designating Controllers, Services as delegators to create the main objects of the core application. The domain layer was centered on representing key concepts for the domain, representing them as entities, aggregate roots and value objects when needed be. Encapsulation of the business core was attained via usage of *DTO* across layers.

HCLC was another one of the guidelines for the development of this application seen as scalability and maintainability are future concerns for the improvement of the concept. As such, an attempt for the lowest possible coupling between layers was thought at all times.

Naming Rules were defined by the team and followed within all the project:

- Camel Case to name all the components in the code.
- Domain interfaces always start with "I".
- Descriptive names to identify the concepts.

Frontend Architectural Design:

For the **Frontend**, the team opted to use **HTML** pages with the help of **Bootstrap** styling, and **Vanilla JavaScript** for the functions and communication with the backend**.**

The frontend is quite self-explanatory. It was designed with simple user experience in mind and to be as intuitive as it could be. As we have little to no experience with frontend development, we understand that many flaws may exist but, with the documentation attached to the home user (the glossary) we believe that it is easy to navigate, and it does all the functions that it is supposed to.

Introduction

User Stories and division choices

In the second sprint, the group identified **7 User Stories** as follow:

US 1 - As an administrator I want to be able to consult, create and delete blacklist items.

US 2 - As a user I want to create a text language analysis by providing an URL (which points to a .txt file), a category and a timeout.

US 3 - As a user, I want to obtain information about the tasks in progress and already completed, by category and/or status.

US 4 - As a user, I want to cancel a task that is currently being processed

US 5 - As an administrator I want to be able to consult, create and delete categories that do not have tasks associated.

US 6 - As a user I want to use the application through a webpage with a user interface.

US 7 - As an administrator I want to use the application through a webpage with a user interface.

But, and although we subdivided them in tickets, we later understood that said division was not the most correct one as, the first 5 should be even more fragmented and, the last 2, were part of all of the above. Which means that, in thesis, we were left with 5 main user stories.

That being said, in this sprint there were many things to correct and update from the backend, mainly refactors and reorganization of the structure and methods that were not as good as we would like to. For that reason, the team splat it in to 3 main areas: Spring Security (users and roles), corrections on what was previously done, and frontend. Please check the *Task Division Chart* to understand how the tasks were divided within the team.

Also, to observe the *User Stories Documentation* updates you can consult the correspondent US Documentation.

Implementation Design Choices

BlackList

In the BlackListItem we used the Value Object BlackListUrl in order to simultaneously make use of java.net URL (for http protocol validation, amongst others) and to implement the value object interface methods, which could not be otherwise implemented if we did not have a wrapper class for it. As it seems, the java.net library is read only and cannot be edited.

Those decisions were supported by the fact that the url is what is used as an identity for the BlackListItem.

Categories

There were 5 categories that were determined by the client to be base categories. Those categories could not be erased nor changed, and one of them had to be determined as the default category when not chosen by the user. Although it was not asked to have the possibility of adding more base categories, the team settled having a easy way to implement it without having to drastically change the domain.

Language Analyzer

The team went for the idea of having an interface to incapsulate business logic of Language Analyzer in a way that it could be easily changed for another analysis API.

Task

As we do not store texts, and to prevent a system overload, we made a verification that, if a certain url was inserted and it is still in process, another repeated one could not be placed before the task was concluded or canceled.

Text

The text is what is analyzed to detect a language. Although there were no specific directions and few rules, we decided to implement some things that we considered to be logic and important as follow:

Blank or empty:

The text has a validation that, if it is a blank or empty .txt file, a correspondent message is delivered to indicate that it was not possible to analyze.

Cleanup:

A cleanUp() method was added to make the tokenization more effective, replacing for a single space all special characters, numbers, empty spaces, multiple spaces and non-Latin alphabet characters.

Hits limitation:

After some tests we have noticed that small texts are not easily identified. More than that, if a language is not valid (without enough hits - words) the task would stay indefinitely in processing status. For that reason, we decided to insert a hits minimum limitation of 2 so we can avoid some of those problems.

Size:

We also decided to have a 5 million maximum limit of characters for performance purposes. As we have a timeout limit of 5 minutes and because we could not effectively kill a thread, we chose a character limit to reduce system and hardware resource usage.

Transient:

We opted to not store in the database the text so we could also spare resources and because we determined that it was not the purpose of this application. In the future it could possibly be stored in order to prevent any analysis to be repeated with the exact same text.

Users

It was asked for the application to have 2 types of roles. One with administrator permissions (being able to manage blacklist entries and categories) and other with permission to create a task to analyze a text language, to cancel a processing task or to consult their tasks.

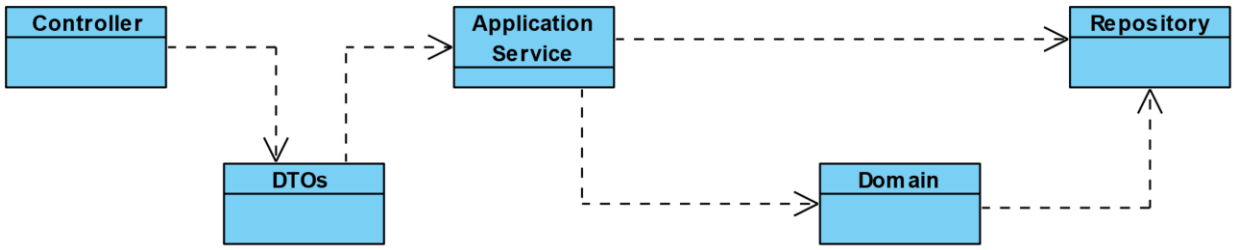
Either way, we implemented it with the possibility of adding new roles and even users with more than one role.

Overview of the application

To better understand how the application is working we will show some of the general diagrams.

Please bear in mind that the diagrams are available in SVG format in the GitHub Documentation.

Design Architecture Diagram

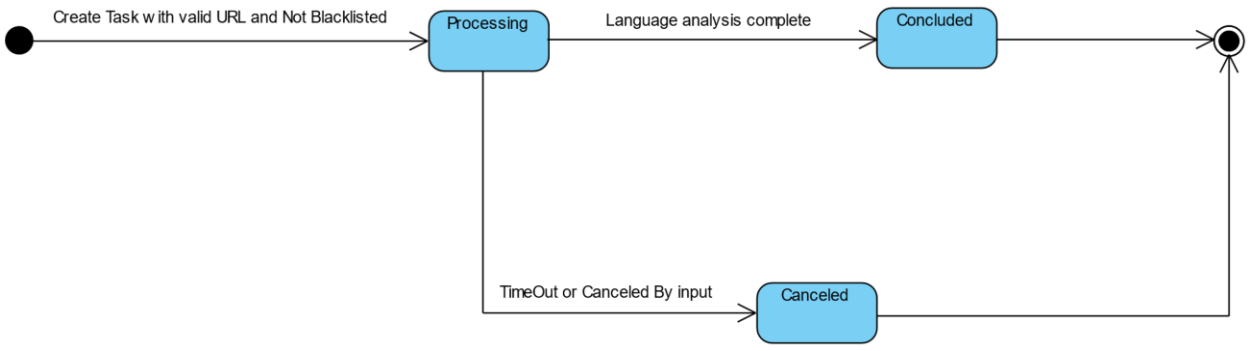


As we can observe above, we opted to have Controllers that will direct the requests to be transformed into DTOs objects to safely deliver it to the application service.

The Application Services are responsible to indicate which process should happen in the Domain. It also communicates with the Repositories that, by their turn, make the bridge with the Database.

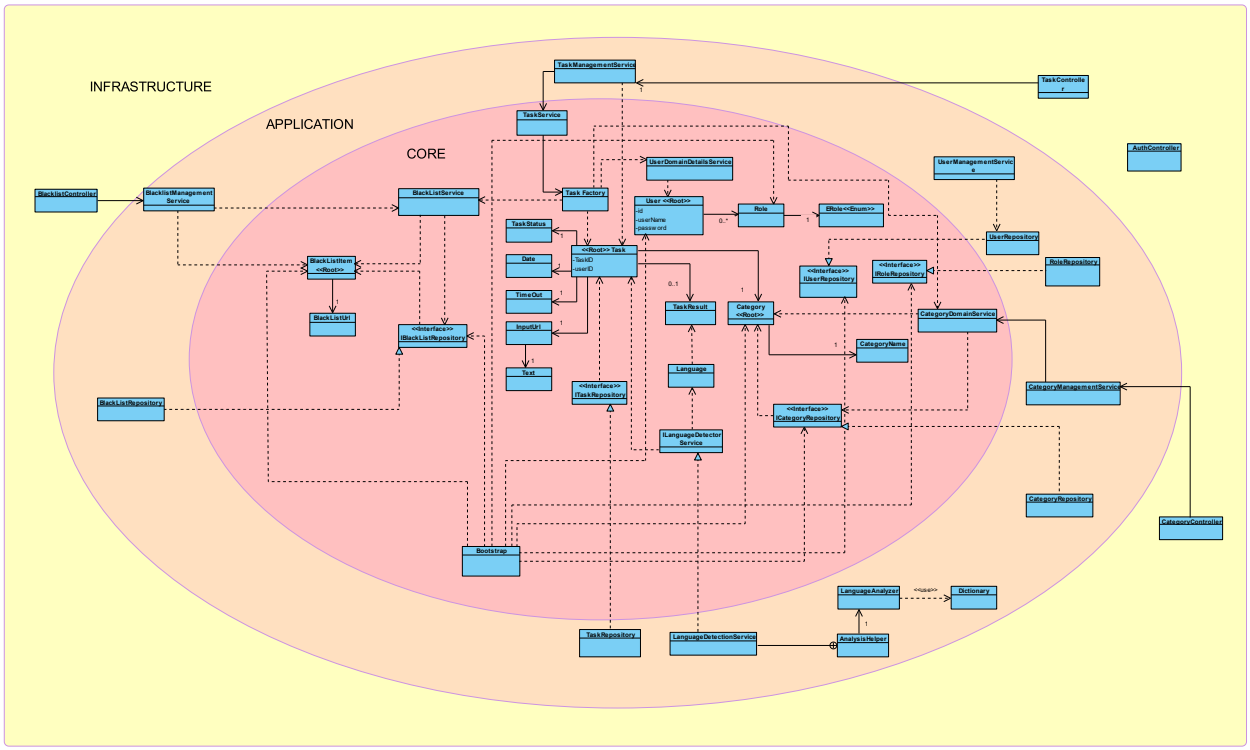
This is a general Class Diagram with all the classes and methods that are involved in the application.

Status Diagram

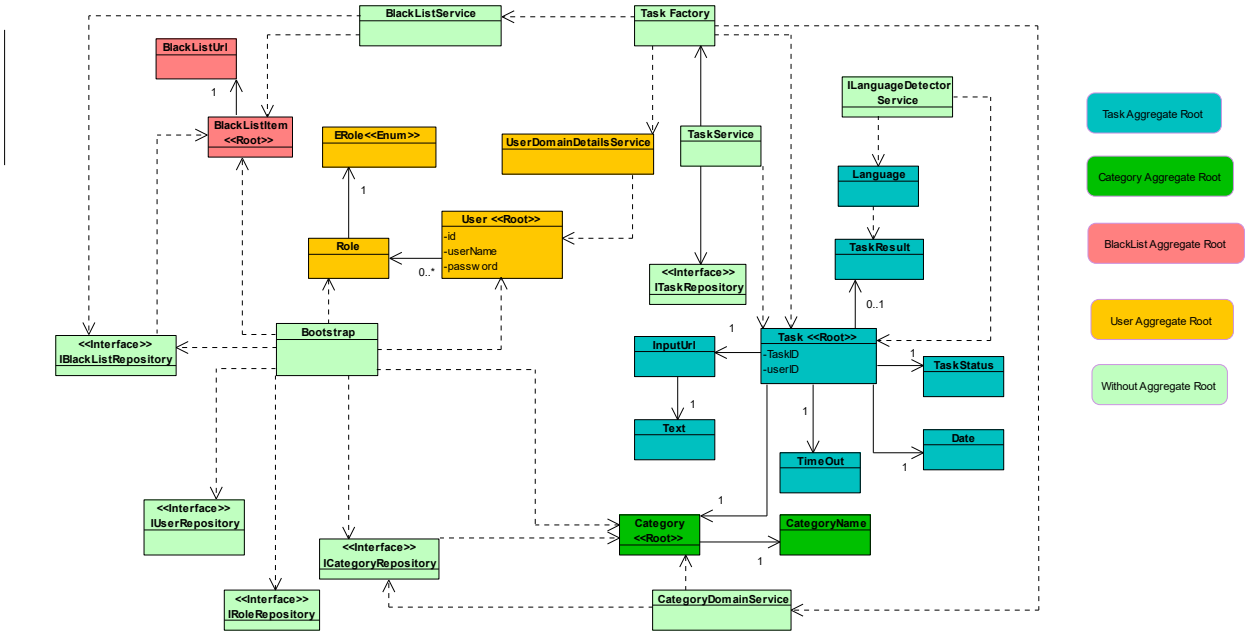


This diagram shows how the status of a task can change between directrices.

Design Layered Diagram

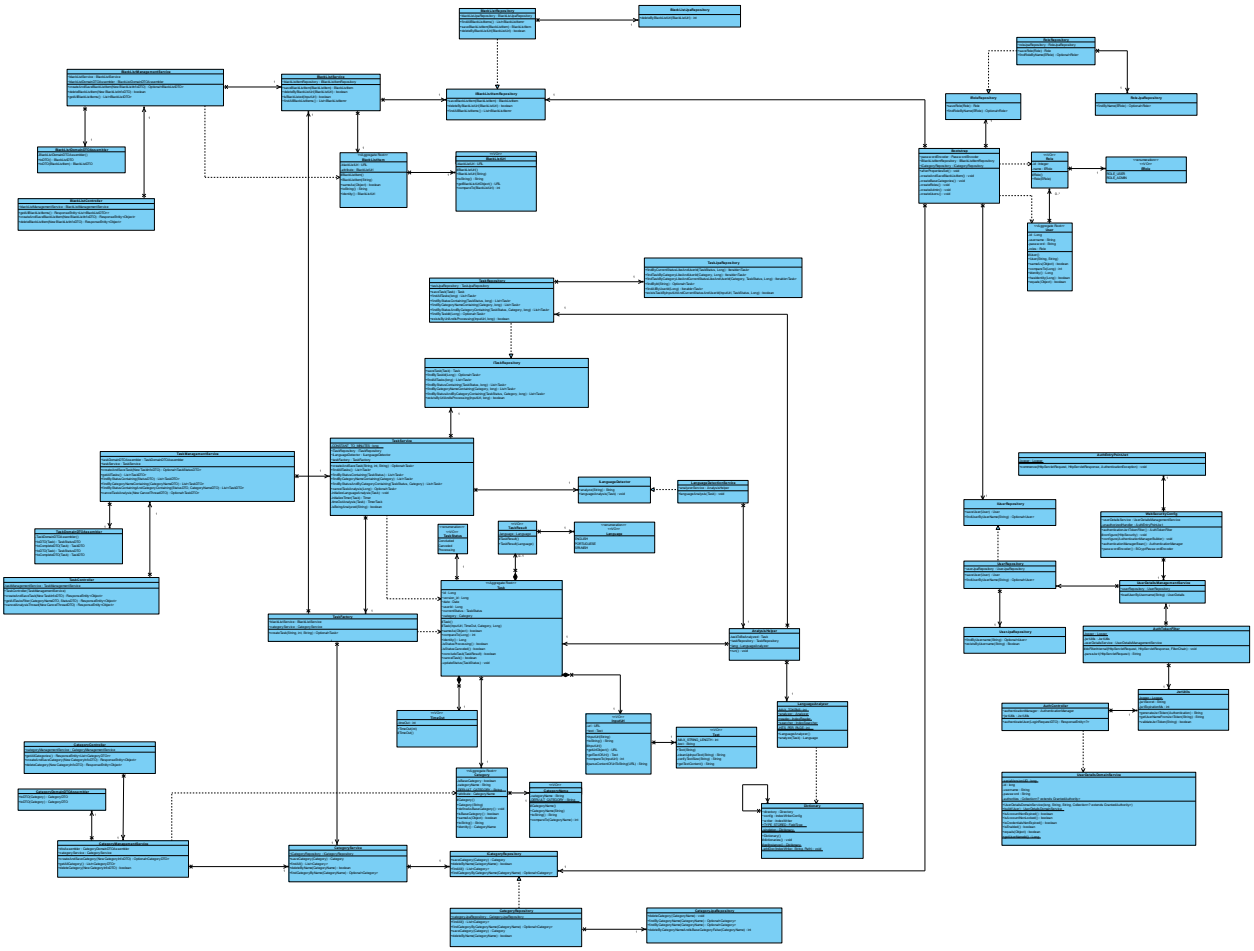


Domain Model



This model is the representation of the main classes involved in the business logic and some of the classes that we considered to be the most important to understand how the layers are being applied.

Class Diagram



This is a general Class Diagram with all the classes and methods that are involved in the application.

Notes

The DTO's are not represented as we thought they would make the diagram even bigger ad therefore even more confused.

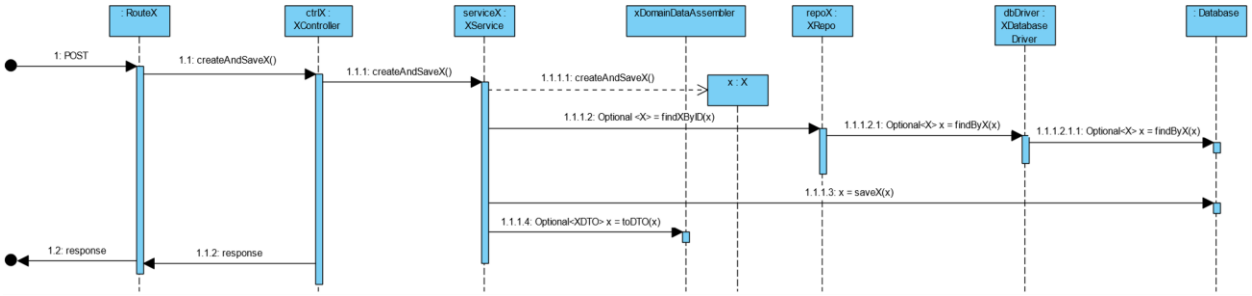
Tags in methods and attributes:

- **+** public
- **#** protected
- **-** private
- **~** package protected

General Sequence Diagrams

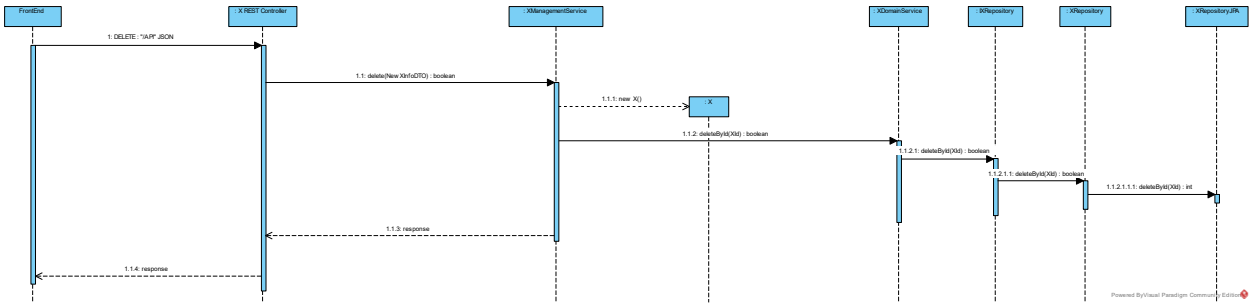
The following diagrams are meant to give an overview of the path of the communication between classes so that the requested processes can happen.

Create (it serves both BlackList and Category)

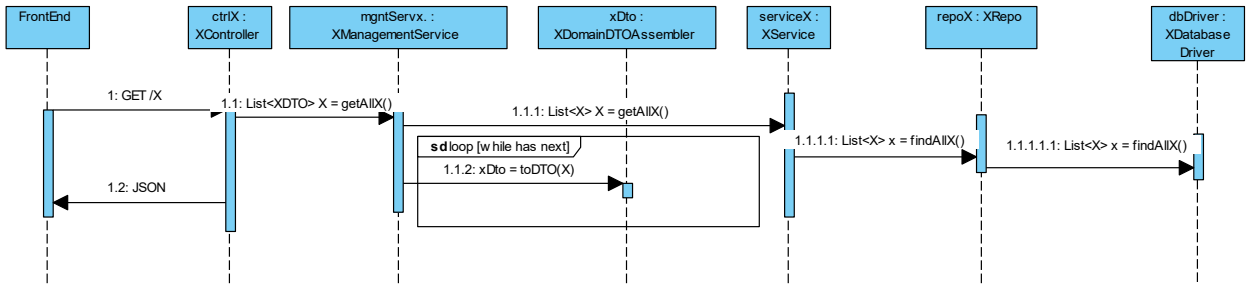


Note: the task creation, as it is much more complex, is presented in the correspondent US document.

Delete (it serves both BlackList and Category)



Find All elements (it serves both BlackList and Category)



Nice to have

Successfully kill a thread:

As we faced several problems when trying to destroy an ongoing thread, and after a lot of study and unsatisfiable answers that it was not possible, we opted to modify the object in the database and return a canceled state to the user as per request without actually killing the thread. An abrupt interruption would generate many complications throughout the whole application. Nonetheless, we are aware that it would be ideal to spare hardware resources and improve overall system performance.

Performance Optimization:

We know some of the flaws of our application. Meaning that, in a task creation for example, we would prefer to have no possibility of repeating the analysis of certain text.

Logout function:

In the frontend API we would like that, when pressing the logout button, the token was killed by consequence and not simply redirect it to the login page.

Response status:

More exceptions with better personalized adequate messages are something that we are lacking, and we would like to have more time to do it so.

Exceptions:

Not every exception is treated in the best way possible. It would be something to be fixed with more time.

Difficulties and Constraints

As always, *time management* can be a problem, especially when we are still learning how to do most of the things, whether in the code, whether in the organization of the overall project.

We thought as well that working with *Lucene* was not easy and turned out to be a challenge in many ways. First, it is not the easiest or most updated tool that Java provides. Secondly, it presented some constraints when working with multi-threading.

In this last sprint we found it especially difficult to *organize* ourselves as all that we had to do was completely new. We had to use Spring Security in order to define roles and there was not any context behind it. For the Frontend, our knowledge in design was little to none and not at all in JavaScript. Everything was a huge challenge that we had to overcome, and it was only possible to implement with a lot of effort, sweat and tears.

We would liked to have more time to better understand *JavaScript* and make a better *Frontend Webpage*.

There were obviously some flaws in the *User Stories* definition and consequently in the tickets' distributions. Although we splat the work equally between team members, we are aware that it might not have been in the best way of doing it so.

About the *Unit Tests*, there were a lot of difficulties felt by the group. As we implemented some features that we did not have the needed amount of knowledge to fully understand (for example Spring Security), some of which were not tested. With Spring Security the only verifications made were through the frontend webpage by asserting that each role only has its authorizations and that a successfully login with the credentials previously created. The other verification that we were able to check was that the token was correctly received in the frontend and updated for each user.

For *Integration Tests*, once again, we were not able to implement it by lack of knowledge. We would love to have had more time to learn more about it.

Another thing that we saw as a challenge was the between *sprints changes* on how we saw the project, and the added difficulty when having to implement or think about things that we did not learn during the course. We know that those challenges reflect quite a lot of what happens in the real business world and, if by one hand it has really frustrated us, on the other hand emphasized our ability to adapt and learn as best and fast as possible.

Speaking about *frustrations* we, as diverse and perfectionist group, had our difficulties when making hard decisions within the project. That was also a learning process that, again, had prepared us for the future. Furthermore, we think that we have a great team, and we could work with each other quite smoothly, with all the respect for one to another and mutual help always in mind.