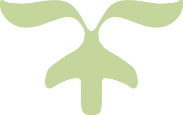


Project Report Analysis

UpSkill Java Capgemini



A picture containing cake, decorated

Description automatically generated

Daniel Machado

Daniel Lima

Inês Clavel

Thales Lemos

SPRINT 2

BY DUCK SQUAD

DUCK SOFT WORKS & CO.

Index

Contents

[Project Requirements Description 2](#_Toc105965174)

[Design choices 3](#_Toc105965175)

[User Stories and division choices 4](#_Toc105965176)

[US1 4](#_Toc105965177)

[US2 4](#_Toc105965178)

[US3 4](#_Toc105965179)

[US4 5](#_Toc105965180)

[US5 5](#_Toc105965181)

[Overview of the application 6](#_Toc105965182)

[Design Architecture Diagram 6](#_Toc105965183)

[Domain Model 6](#_Toc105965184)

[Class Diagram 7](#_Toc105965185)

[Status Diagram 7](#_Toc105965186)

[General Sequence Diagrams 8](#_Toc105965187)

[Create (it serves both BlackList and Category) 8](#_Toc105965188)

[Delete (it serves both BlackList and Category) 8](#_Toc105965190)

[Find All elements (it serves both BlackList and Category) 8](#_Toc105965192)

[Nice to have 9](#_Toc105965194)

[Difficulties and Constraints 9](#_Toc105965195)

# 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

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.

# User Stories and division choices

For the general application, 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.

For this sprint we decided to focus on the **first 5 User Stories** and we divide them in smaller **“use cases” or tickets** to be split between the team as equally as possible.

### US1

1. Create a blacklist entry with persistence
2. Consult all blacklist entries
3. Remove a blacklist entry
4. Insert the blacklist entry, given by the client by bootstrap

### US2

1. Create an analysis task, with a category, a timeout and with persistence that saves a status, an URL, and a category
2. Confirm that a URL is or not in the blacklist
3. Make sure that the URLs inserted are in .TXT file type.
4. Check if the category passed by the user exists
5. Accept and validate a time limit for the analysis task
6. Start analysis task asynchronously

### US3

1. Consult tasks by status
2. Consult tasks by category
3. Consult tasks by category and status

### US4

1. Interrupt an ongoing thread
2. Save and persist a cancellation of a task

### US5

1. Create category and persist it
2. Delete categories from the database
3. Consult categories created in the database
4. Bootstrap of the 5 base-done categories
5. Prevent the 5 base categories from being eliminated

To see the *tickets division* please consult the “Tickets Chart” that includes the division per teammate, the effort scale of each ticket and the time duration of each one.

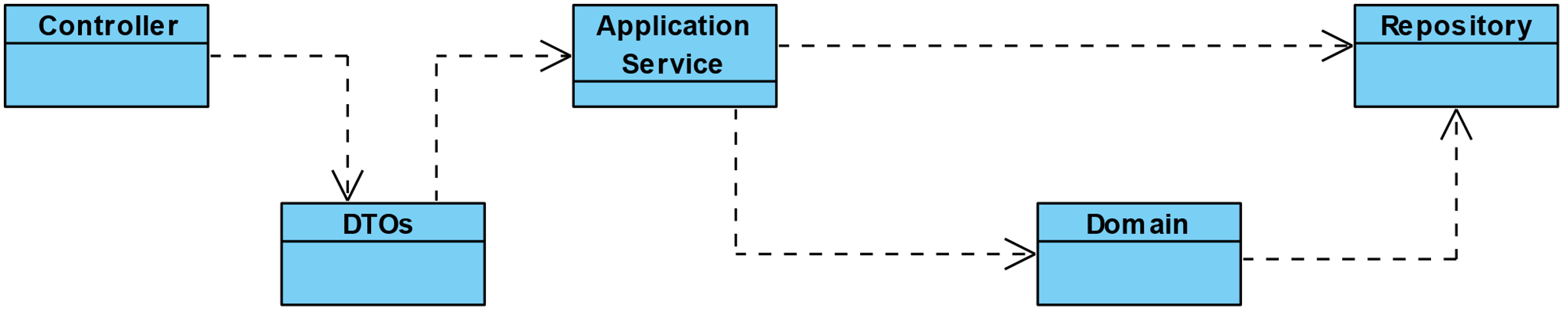
Also, to observe the *User Stories Documentation* in detail please consult the correspondent US Documentation.

# 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.

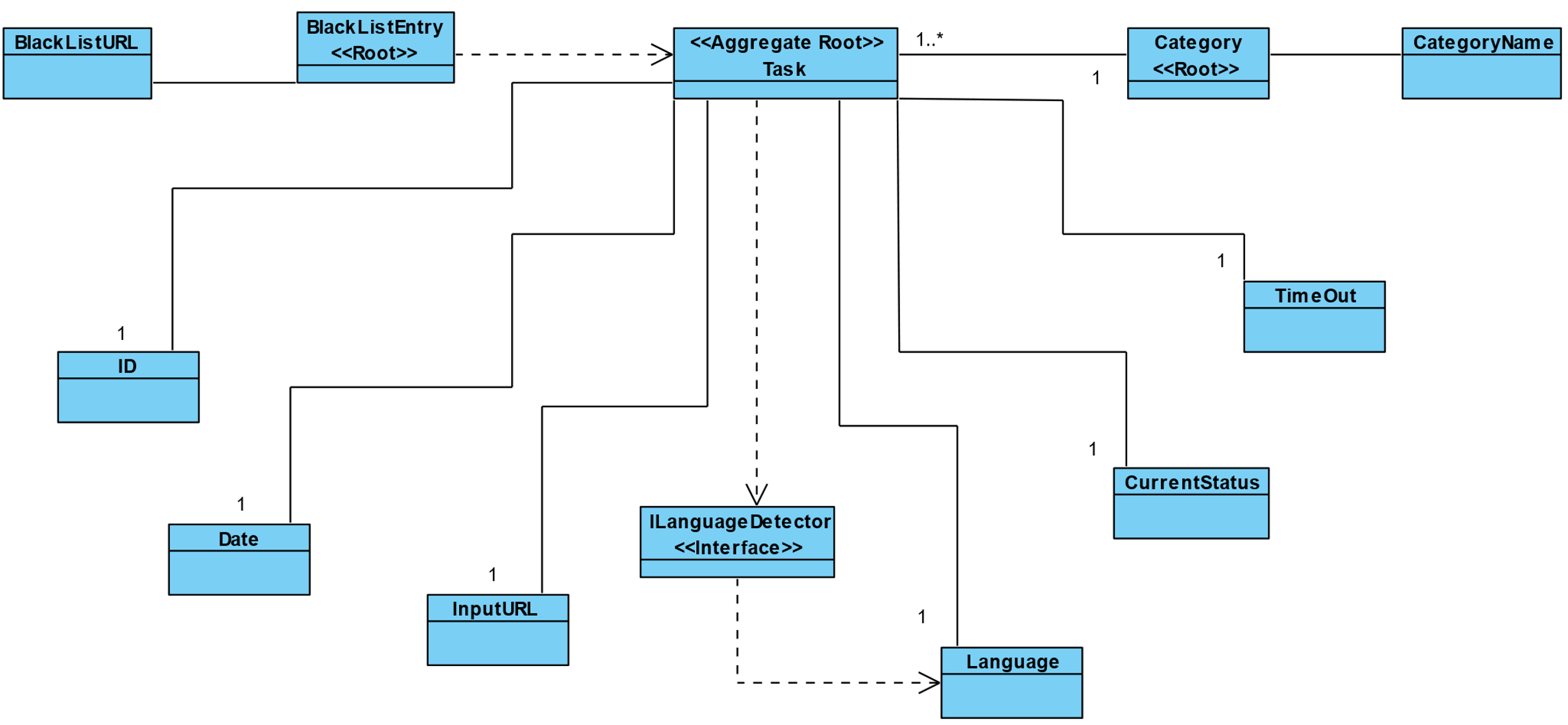
## Design Architecture Diagram



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.

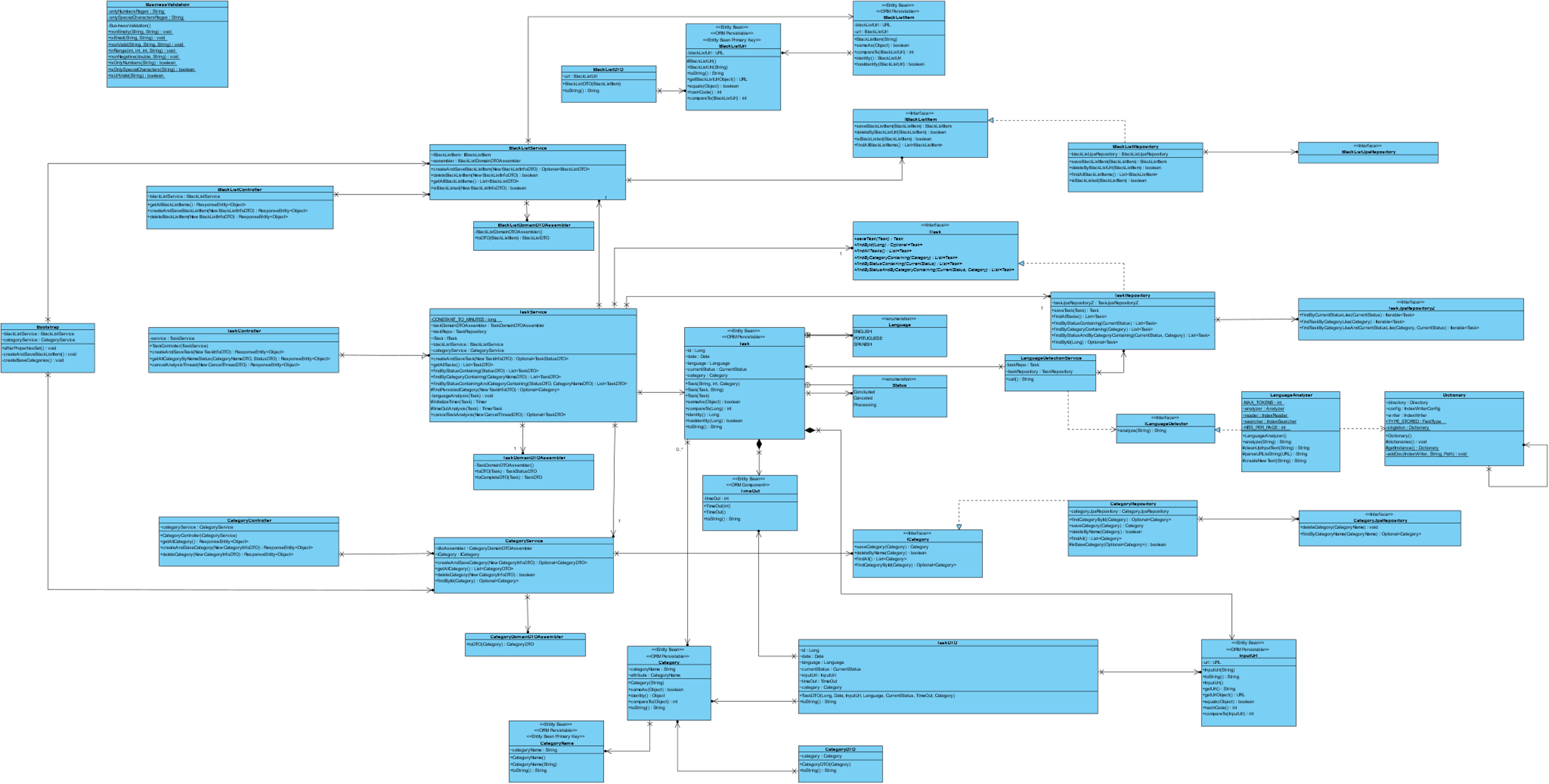
The Application Services are responsible to indicate which process should happen in the Domain. It also communicates with the Repositories that, by their turn, make the bridge with the Database.

## Domain Model



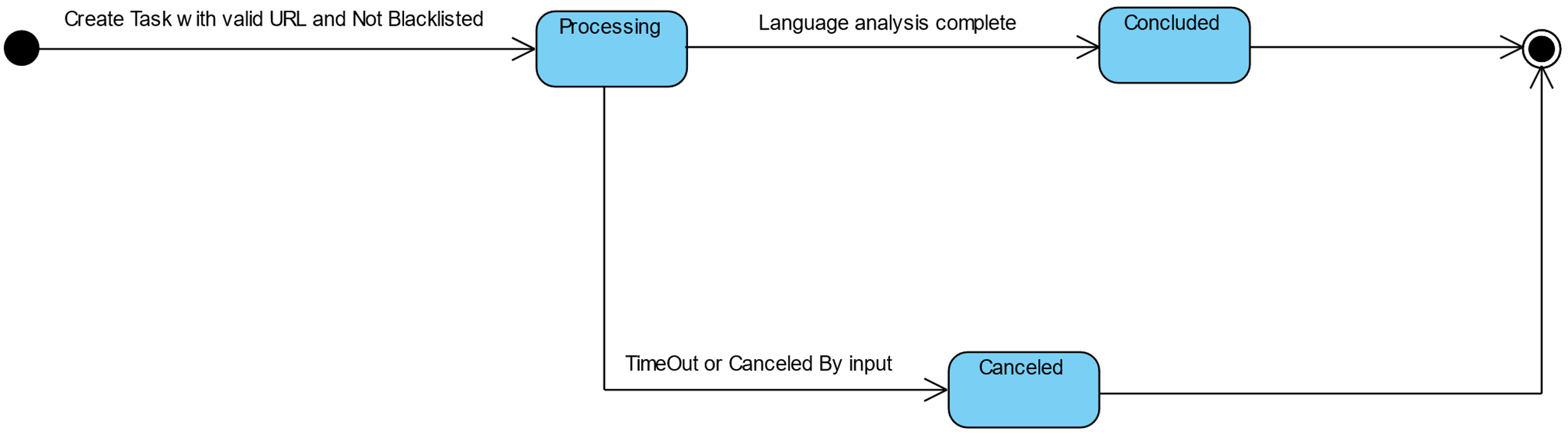
This model is the representation of the main classes involved in the business logic.

## Class Diagram



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.

# 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

As we faced several problems when trying to cancel 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 with 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.

Also, we would like to have a system prepared to reject a task processing request, when the url contained within had already been attributed with a language. This would be more efficient as the purpose of the application is to determine a language , and in this scenario, the criteria would’ve already been met. In addition to this , there is the possibility of over encumbering the system with multiple requests for determining a language for the same url.

In order to do so, however, it would be necessary that the text associated with the mentioned url had previously persisted, which, by not being an explicit requirement, could even mean contradicting the client’s demands.

# Difficulties and Constraints

As always, time management can be a problem, specially 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.

Another thing that we saw as a challenge was the between sprints change of the whole project, and the added difficulty when having things that we did not learn during the course. We are aware that those challenges kind of reflect a lot of what happens in the real business world and, if by one hand it really frustrated us, on the other hand emphasized our ability to adapt and learn as best and as fast as possible.

Speaking about frustrations, we are dealing constantly with the frustration of not having things working as well as we would like to when being a perfectionist group. That is also a learning process that, again, is preparing us for the future.

As for working as a team, it also can be a challenge, but as a group we think that we have a great team and we can work with each other quite smoothly, with all the respect for one to another and mutual help always in mind.

For this sprint, we consider that we do almost exactly what we had planned to do. Of course, there are many adjustments to make, but not everything can be perfect. Maybe next sprint.