2DV603 - Software Engineering

Thesis Management System

Design Document

Author: Ahmad Abdilrahim

Author: Ahmad Sadia

Author: Caesar Alhawi

Author: Cristian Babes

Author: Marcello Vendruscolo

Supervisor: Mirko D'Angelo

Supervisor: Mauro Caporuscio

Semester: VT 2019

Area: Computer Science

Table of Contents

1 – Introduction

The first section of this document explains the document structure, its purpose, and references back to the requirements document. It provides the reasoning for the need of such a document and which requirements this document addresses.

1.1 – Purpose of the Design Document

The main purpose of this document, also known as software design document (SDD) or technical specification, is to provide information regarding the technical approaches chosen in order to implement a software system. In order to achieve its goal, the design document is structured in a manner that its first chapters present a broad overview while the last section presents the design details on the implementation level. The importance of such a document is to ensure that, during the implementation stage, the development team is synchronized, the correct work gets done, and that, consequently, the application achieves the expected high-standard software quality. Furthermore, the importance of a design document goes beyond the previously mentioned reasons. It encourages the involved software engineers to reflect about the application and to collect feedback from their colleagues. The information provided by such document should be detailed enough to allow the development team to proceed with a good understanding of the application to be implemented and the plan to be followed. This design document provides the design details of the thesis management application developed for the computer science department of Linnaeus University in Sweden. The target audience of this document are the project managers, a range of university employees related to IT-technical support, and, finally, the development and maintenance team.

1.2 – Requirements Traceability

This section indicates which requirements from the requirements document are implemented and design detailed. The following list contains all the requirements previously identified, but since some of them are not implemented in the application, they are in red font-color.

F.R. 1 - Users shall be able to log-in the application
F.R. 2 - Users shall be able to log-out the application
F.R. 3 - Students shall be able to create submissions
F.R. 4 - Students shall be able to read submissions
F.R. 5 - Students shall be able to update submissions
F.R. 6 - Students shall be able to delete submissions
F.R. 7 - Students shall be able to view a list of supervisors
F.R. 8 - Students shall be able to suggest a supervisor
F.R. 9 - Students shall be able to view the content of feedbacks
F.R. 10 - Students shall be able to view their thesis final grade
F.R. 11 - The coordinator shall be able to view a list of all currently registered users
F.R. 12 - The coordinator shall be able to assign the supervisor role to users
F.R. 13 - The coordinator shall be able to view a list of students assigned to supervisors
F.R. 14 - The coordinator shall be able to view all submissions
F.R. 15 - The coordinator shall be able to evaluate project description submissions
F.R. 16 - The coordinator shall be able to evaluate thesis submissions
F.R. 17 - The coordinator shall be able to evaluate final thesis submissions
F.R. 18 - The coordinator shall be able to set deadlines
F.R. 19 - The coordinator shall be able to update deadlines
F.R. 20 - The coordinator shall be able to submit the grade for a thesis project
F.R. 21 - Supervisors shall be able to confirm supervision requests
F.R. 22 - Supervisors shall be able to decline supervision requests
F.R. 23 - Supervisors shall be able to assess the students' project plan submissions
F.R. 24 - Supervisors shall be able to assess the students' report submissions
F.R. 25 - Supervisors shall be able to view students' project plan submissions

F.R. 26 - Supervisors shall be able to view students' report submissions

F.R. 27 - Readers shall be able to bid for reports of their preference

F.R. 28 - Readers shall be able to feedback submissions

F.R. 29 - Opponents shall be able to view assigned reports

F.R. 30 - Opponents shall be able to give feedback to an assigned report

F.R. 31 - The coordinator shall be able to view the role assigned to a user

F.R. 32 - The coordinator shall be able to assign the reader role to users

F.R. 33 - The coordinator shall be able to assign the opponent role to users

F.R. 34 - The coordinator shall be able to update the supervisor role of users

F.R. 35 - The coordinator shall be able to update the reader role of users

F.R. 36 - The coordinator shall be able to update the opponent role of users

N.F.R. 1 - Users shall be able to familiarise themselves with the application within 5 minutes

N.F.R. 2 - Users' private data shall be stored and processed in a manner that high security standards and data integrity are priorities

N.F.R. 3 - The log-in functionality shall not be affected by SQL-injection

N.F.R. 4 - The system shall have average response time less than 10 seconds for submitting a document

N.F.R. 5 - The system shall have 100% uptime throughout the course duration

N.F.R. 6 - The system shall comply with the General Data Protection Regulation (GDPR)

N.F.R. 7 - The system shall not depend on the customers operation system and shall support the mainstream web browsers

N.F.R. 8 - The system shall be documented with requirements document, design document, and UML diagrams

N.F.R. 9 - The system shall answer at least 60 concurrent upload requests in less than 10 seconds

N.F.R. 10 - The user interface shall be available both in English and Swedish

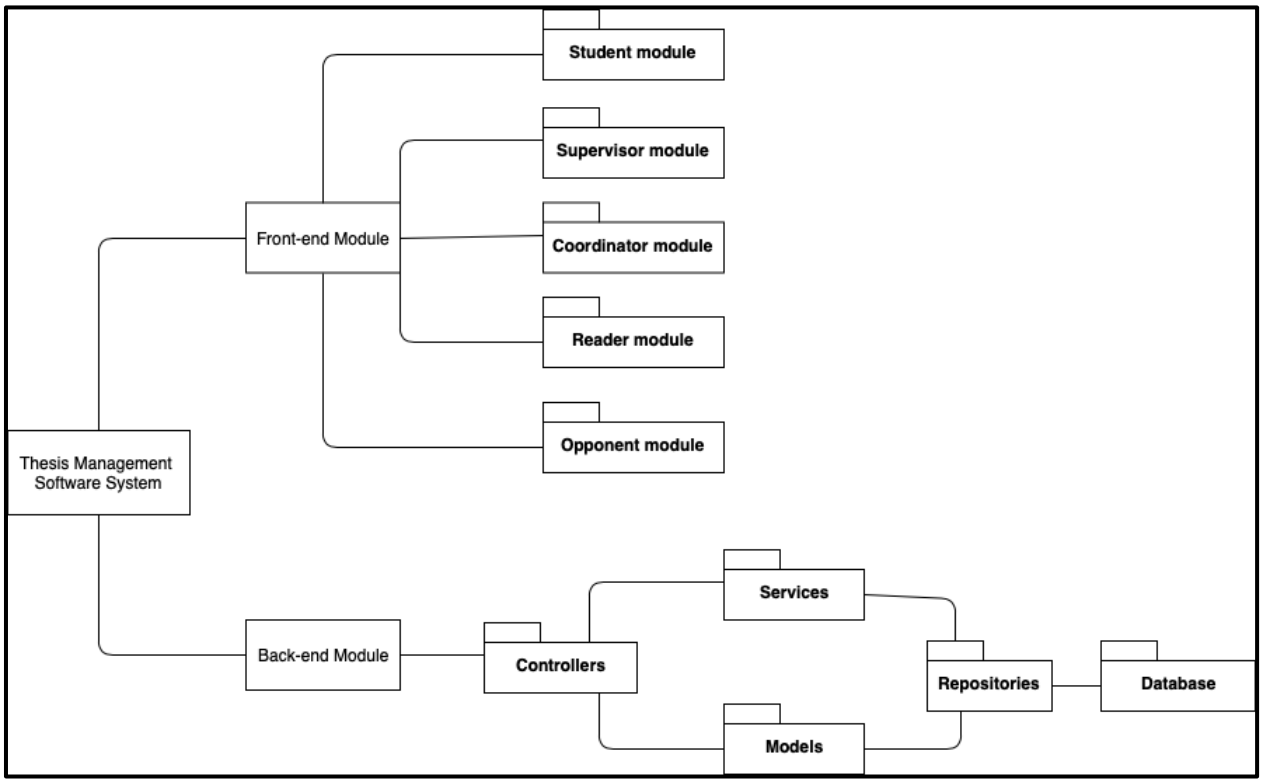
N.F.R. 11 - The system shall accept only .pdf submissions

2 – General Priorities

Throughout the engineering process, several design principles are used as guidance in the decision-making protocols since high standard quality is a priority in this thesis management software. In this chapter, the chosen design priorities that best suit this project are identified along with an explanation why they are prioritized over other principles.

2.1 – Modulatory

This principle is one of the most essential and widely used principles in the software engineering field. It states that systems should be implemented from low coupled, cohesive modules. Essentially, this principle translates the divide and conquer idea, and focuses in breaking down large systems into smaller sub-systems. By following this principle, components are created with well-defined purpose and functions, as well as the dependencies between the components are reduced to minimal. The benefits of following this principle are more flexible, comprehensive components that can be easily maintained, reused and replaced if needed. Moreover, individuals of the development team are able to concurrently work on different parts of the system without affecting for code. Furthermore, modularity also makes concurrent execution of the system possible. These are the reasons why modularity is one of the most important design principles in our application. It was achieved by dividing the application's front-end into student, coordinator, supervisor, reader, and opponent modules, and the back-end into controllers, services, models, repositories modules together with the database. The following image illustrates how the application system is divided into modules.



2.2 – Maintainability

Application software that has high maintainability allows fast, simple, and secure modifications (or replacement) of its modules. It is a characteristic that is embedded in the product's design. On the other hand, an application software that is not designed to be easily maintainable can generate complex tasks to the software maintenance team besides the excessive costs and long out-of-service periods. Therefore, since early in the designing process, the thesis management application embodied this principle as one of the most important principles to be followed. The modularity principle previously described indeed improves maintainability since the modules are loosely coupled and can easily be replaced without carrying extensive changes to the system. Some of the rules followed in order to achieve high maintainability in this software are to keep components' structure simple since complex structures are hard to maintain as well as to make components testable in order to find faults in the system in early stages according to the preventive maintenance guidelines.

2.3 – Abstraction

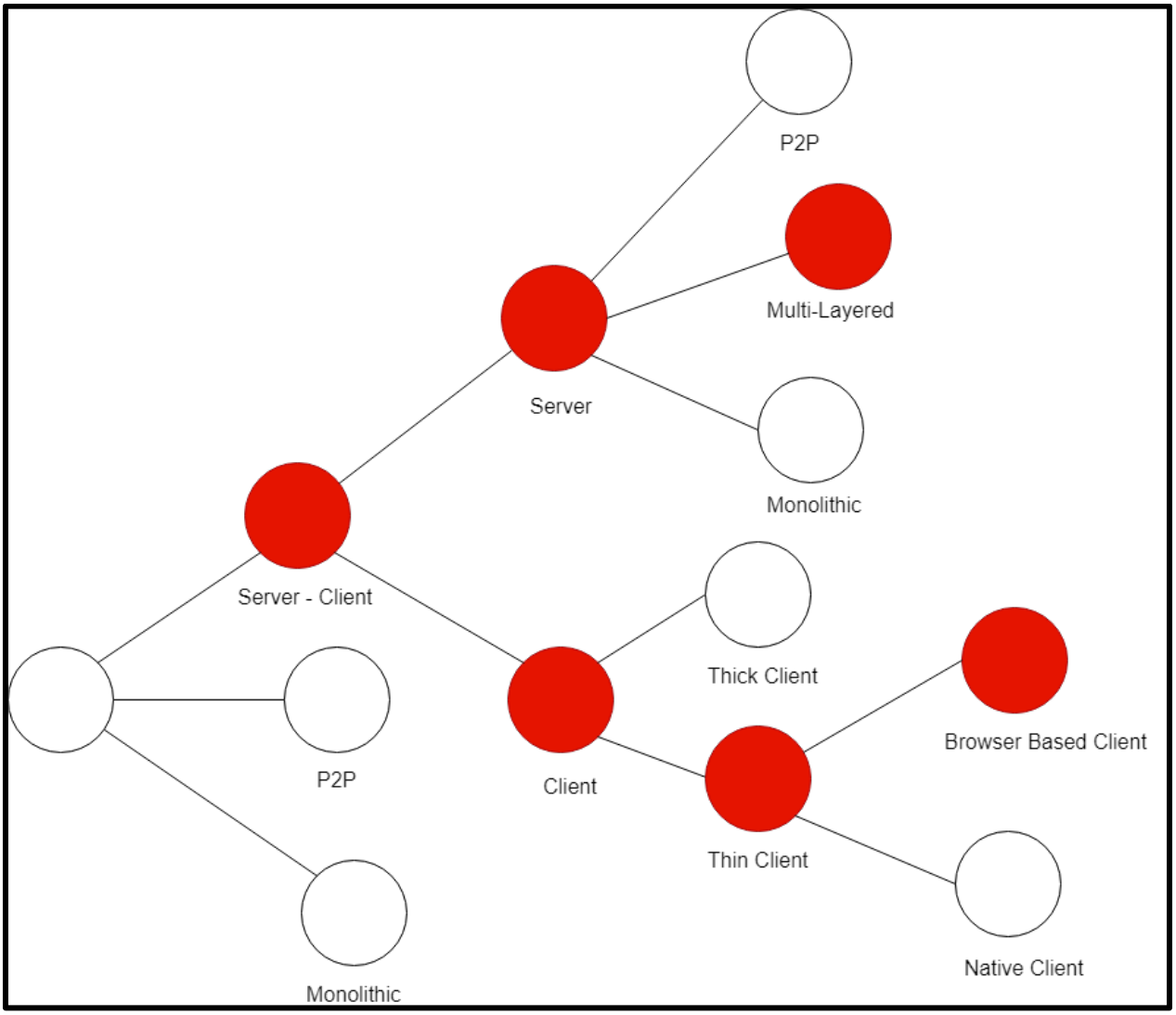
Software systems that are designed for high-level of abstraction are able to successfully hide and, therefore, protect details and attributes of the application system. Moreover, designing for high-level of abstraction generally simplify the overall software structure, and support portability. This thesis-management application, in order to achieve abstraction, is designed in such a manner that control-flow is taken into consideration and the components of system only communicates using well-defined interface and set of instructions.

2.4 – Cohesion

Consequence of the modularity principle implementation is that the software system becomes easier to understand and change. Furthermore, components with high cohesion can easily be reused and replaced. In the thesis management software, the client modules are created and separated based on the target user of their functionality. For example, the functionality for the coordinator role is grouped only with other coordinator functionalities. Moreover, in the back-end side, due to the architecture imposed by the Spring framework, the modules are forced together based on their layers.

3 – Design Overview

In this third section of the document, a high-level description of the decision-making steps guided the general application architecture is presented. Moreover, it also shows the different alternatives that were considered while the design was being decided and the reasoning behind each decision. The following design space shows the set of all possibilities considered at some stage of the design engineering process.



- Server-Client: Since the application is a web application, the HTTP protocol imposes the web application to behave according to the client server architecture. Also, this architecture let the responsibilities and roles of a computing system to be divided among several computers. Thus, it leads the software system to a more well-structured and easier maintenance, manageability and scalability. This means that replacing, editing, or upgrading a client are smoothly can be achieved with this architecture. Also, the data is stored in the server, thus, more security controls than most having them in clients. Furthermore, when the data storage is centralized, updating the data becomes easier to manage, when the storage of the data is centralized. Moreover, the client server architecture selected improves security by ensuring that only authorized clients can access and manipulate data on the servers. Peer 2 Peer was dropped due to its unsuitability for websites. Two or more of the same application acts as server and client at the same time towards each other. Monolithic architecture is very fast but not scalable. That means the system has to be ignited every time a client uses it. It also makes it hard to understand the system or even implement changes in a complex application because of the lack of modularity. Thus, it was not considered for this part of the system.

- Thin client: The decision to choose this for the design was mainly because as we are making a web application, we need the server to do most of the computational tasks. A thick client means that most of the tasks are executed on a local computer which does not server the purpose and functionality of the Thesis Management System very well.

- Multilayered Server: This type of architecture was chosen because it promotes modularity and makes it easier to add new subsystems. Peer 2 Peer architecture was not chosen because it is not suitable for websites, and it is difficult to administer. Monolithic architecture makes the system hard to understand and implementing changes can be challenging in complex systems. It also creates security and access control problems because in order to restrict access to certain data, a permission-based system is used, and these systems can be subject

to hacking or other kinds of data exploitation. Therefore, monolithic was not considered for the server part.

- Browser Based Client: The application is required to be a web application. Therefore, a browser-based client is considered to meet that requirement.

4 – Detailed Designing Choices

This last chapter of the document presents the general architecture of the software, and also the detailed component implementation architectures through unified model language (UML) class diagrams.

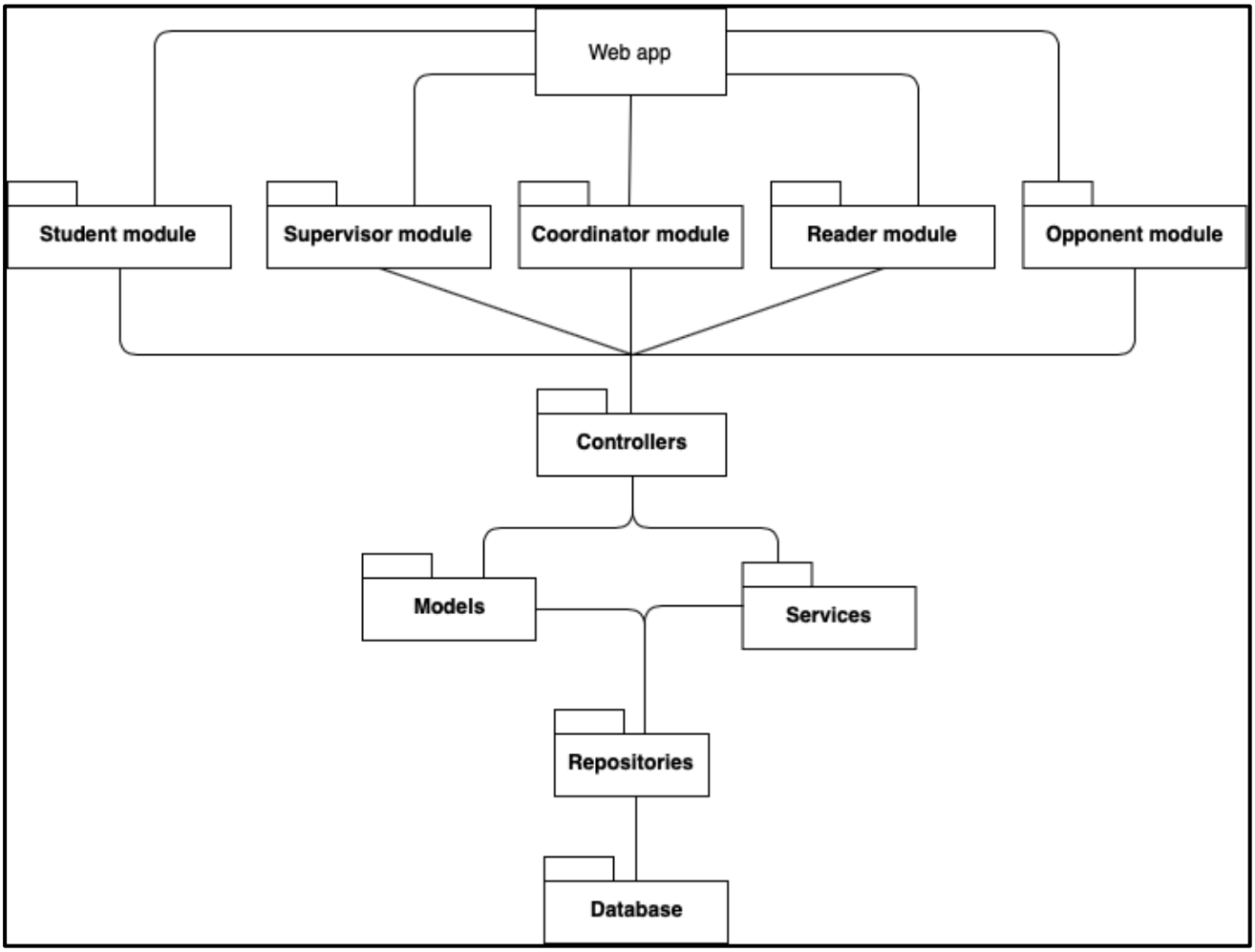
4.1 – Software Architecture

The general architecture of the application was designed taking into consideration all the four previously mentioned design principles, but the modularity principle was always the strongest priority implemented since it has noticeable impacts on the other principles. Moreover, as previously explained, the combination of server-client and layered patterns were chosen as main architectural patterns guiding the designing options. Then, combining the designing principles and the architectural patterns together, the result is a software system architecture that has six main layers:

1. Web app layer which represents the client and holds as little application logic as possible.
2. Subsystem layer which is the first actual layer in the server side. It is composed of different independent subsystems, and each subsystem takes care of different functionalities of the entire system. They are grouped based on the actor they serve, and they are completely disconnected and independent. Therefore, if, one actor is removed, for example, the corresponding subsystem can be safely removed. This characteristic also allows the development team to add new subsystems without extensive modifications.
3. Controllers layer works as a bridge between the front and back-ends. It is the layer which application clients access the services provided by the system.

4. Services (and models) layer contains the API of the most CRUD functionalities provided by the system. Also, this layer works as the layer that has direct access with the repository classes.
5. Repositories layer which is responsible for controlling the data that enters and leaves the database. This database controller layer is composed by smaller components previously mentioned, such as Thus, it acts as an interface/API for the database
6. Database layer which represents where all the information is stored

The general software architecture is described through the following UML component diagram



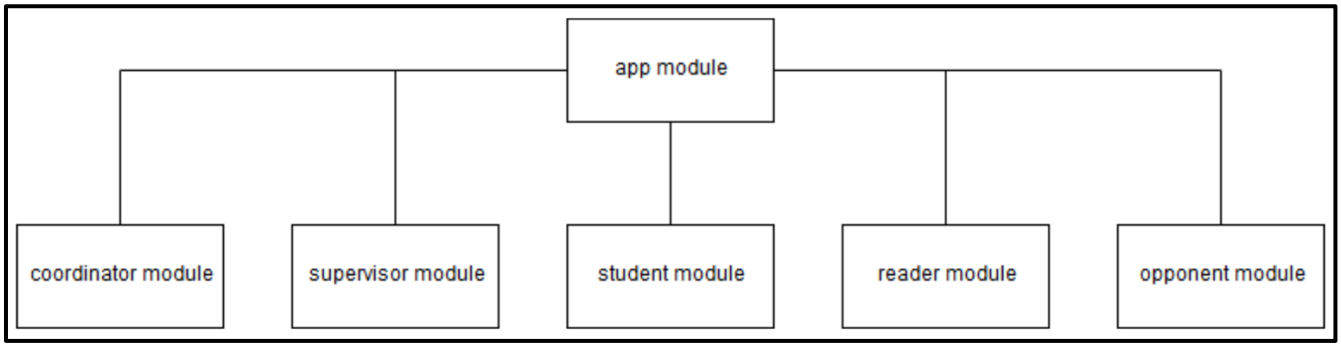
4.2 – Components Design Implementation

This section of the document introduces UML diagrams for each component presented in the application.

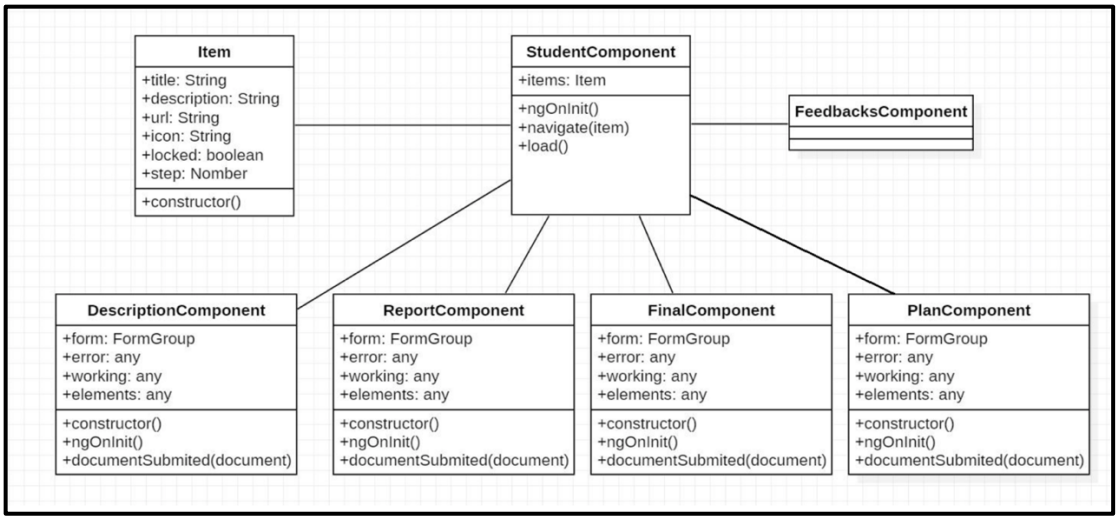
4.2.1 – Front-end Modules

Each of the previously mentioned module in the front-end side of the application has a UML class diagram. The diagrams are presented in order to illustrate how the design principles are, in fact, delivered. First, a package diagram shows the relationship between the module in the front end. It is important to notice that the classes existing in a package does not have direct relationship to the classes in the other packages.

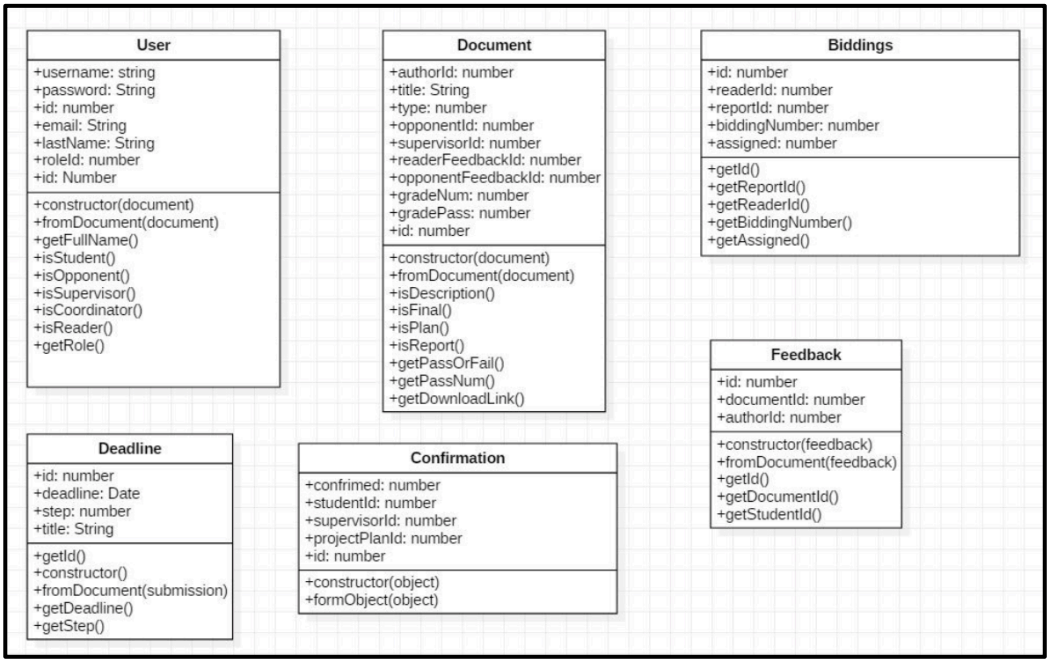
- General modules diagram: This diagram represents a view of the models and how the web app is structured. Each module is presented in detail in separate class diagrams that follows. Notice, this module and the following class diagrams do not represent the systems in terms of dependencies, but rather an abstract structure which corresponds the flow that a user follows when navigating the system. This limitation is due to the manner that the front-end framework used, Angular, is built.



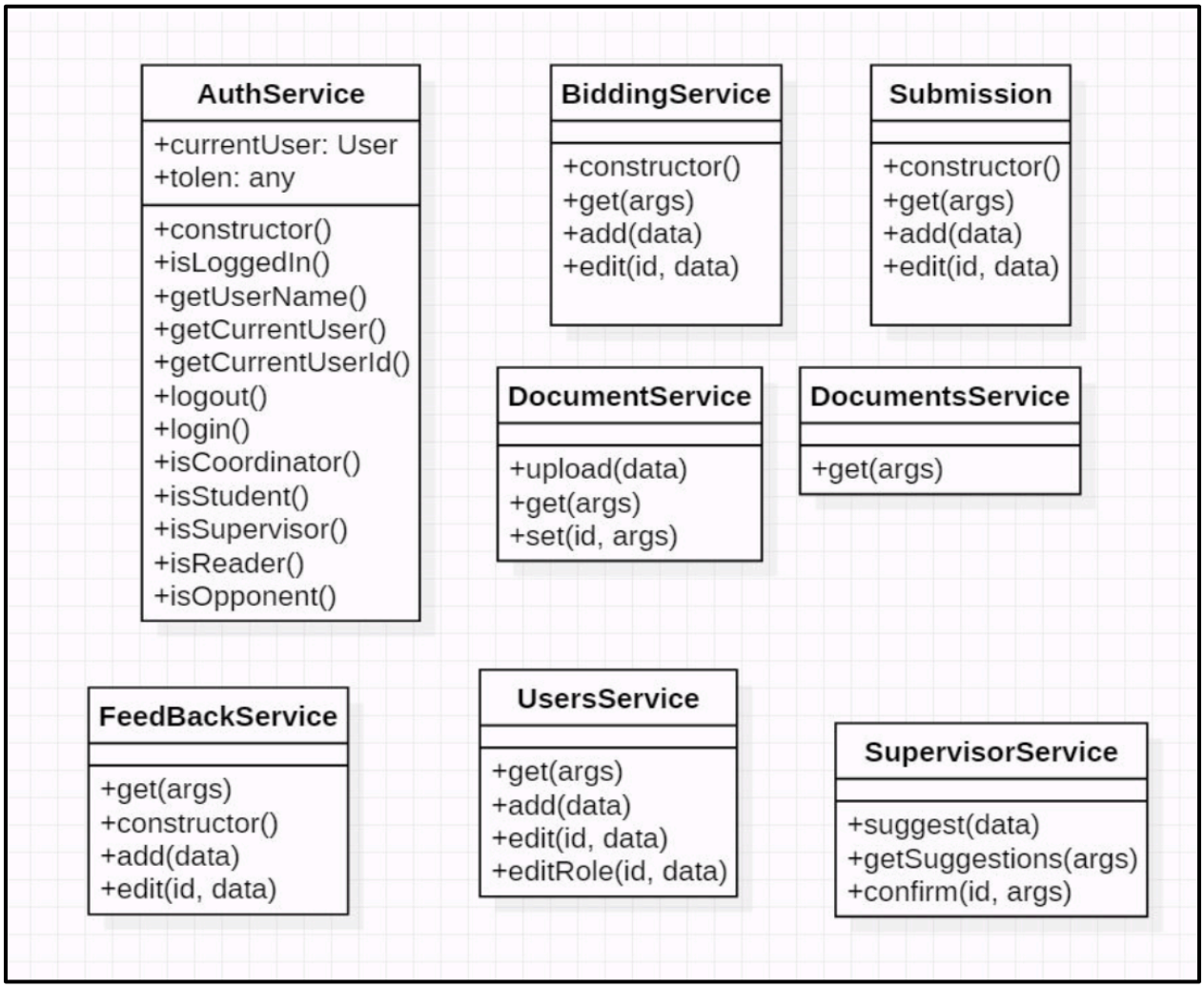
- Student module: The diagram represents the student module. This module is relatively simple, and most functionalities are file uploads.



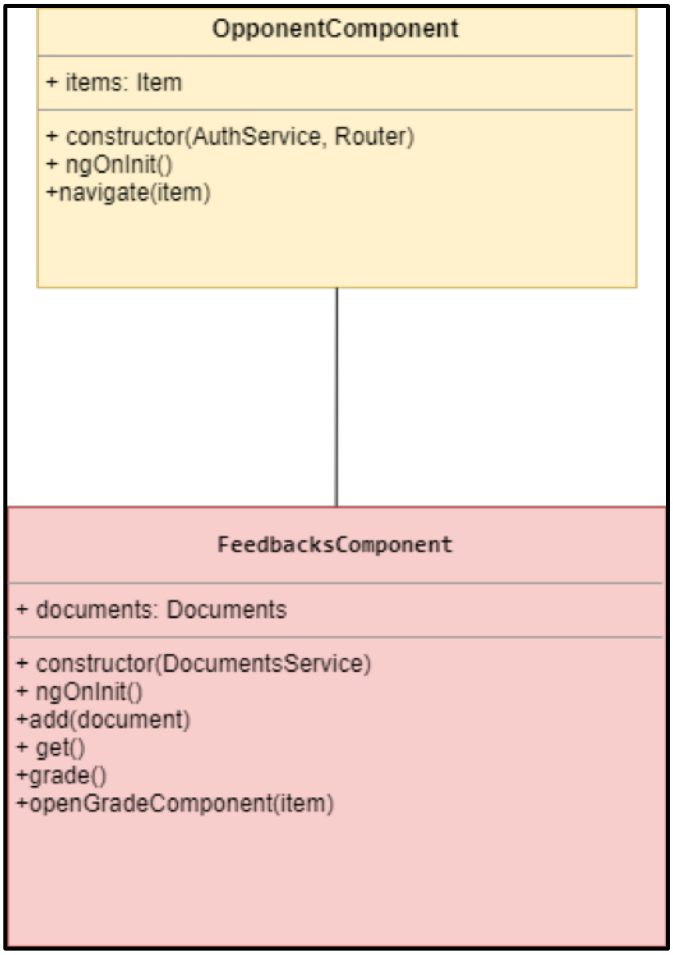
- Modules diagram: These classes represented the entities in the system, In the other diagrams you could see classes that are represented here.



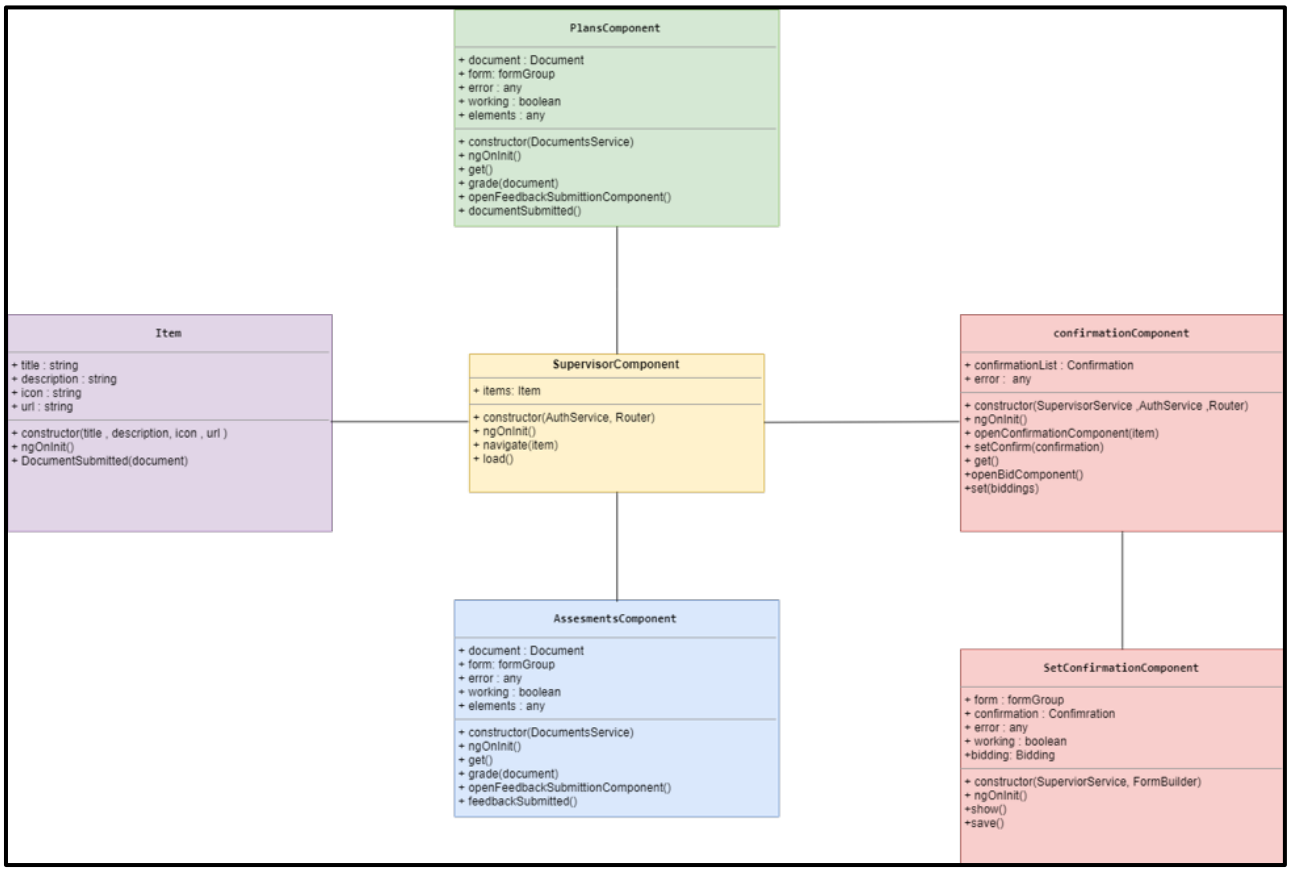
- Service classes diagram: This class diagram represents the services in the system which are used to communicate with the back-end and provide other functionality that is required in more than one component. They are not modelled in other types of UML diagrams because of the manner that Angular handles their injection.



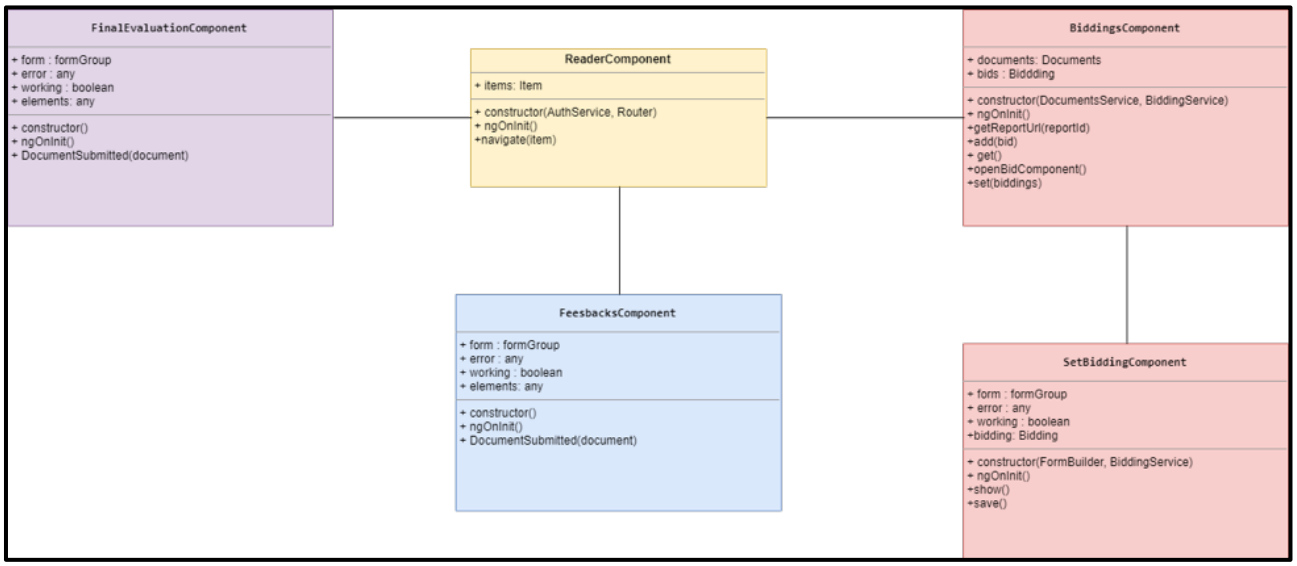
- Opponent module diagram: The following diagrams represents the opponent component. When a user logs in they are represented with a feedback option. The opponent component has a sub component which is the feedbacks component which gives the user the option to download and submit a feedback for the report assigned to them.



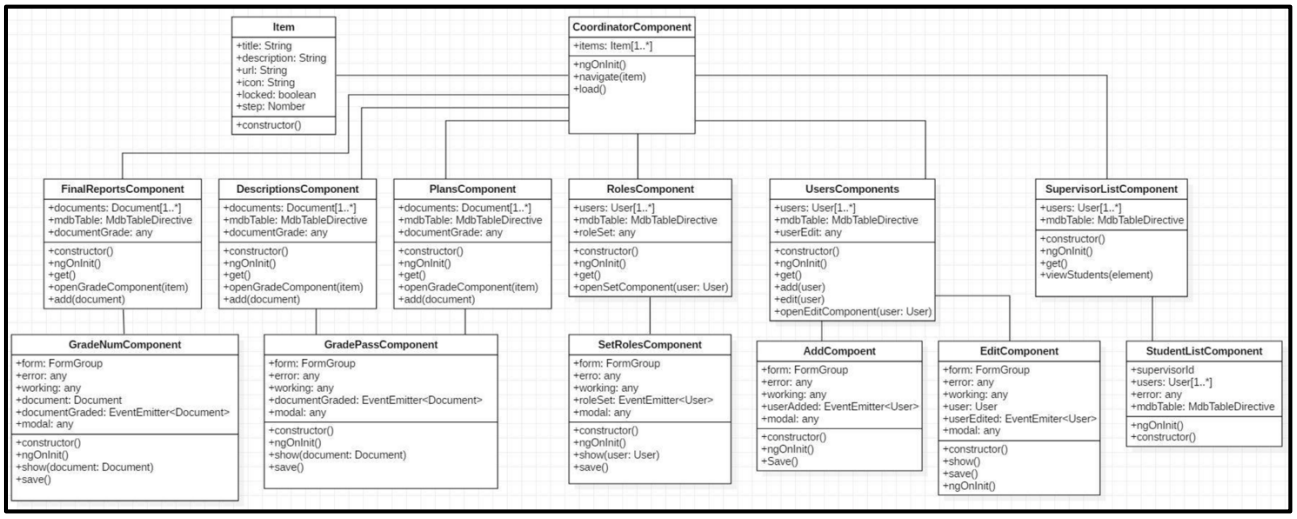
- Supervisor module diagram: The following diagram represents the supervisor component. The following component has several sub components which represents different options for the supervisor. The plans component gives the supervisor the options to review and give feedback for project plans. The assessments component represents the to assess report for students assigned to a specific supervisor. The confirmation component presets the supervisor with a list of students who choose the supervisor and the ability to confirm each student supervision.



- Reader module diagram: The following diagram represents the reader component. the component has three different sub-components. When a reader logs in they are represented with three option. The bidding component allows readers to bid of reports, the feedbacks component allows the option to submit a feedback for the report which a reader have bidden and lastly final evaluation component which allows the reader to submit a feedback for the final report.



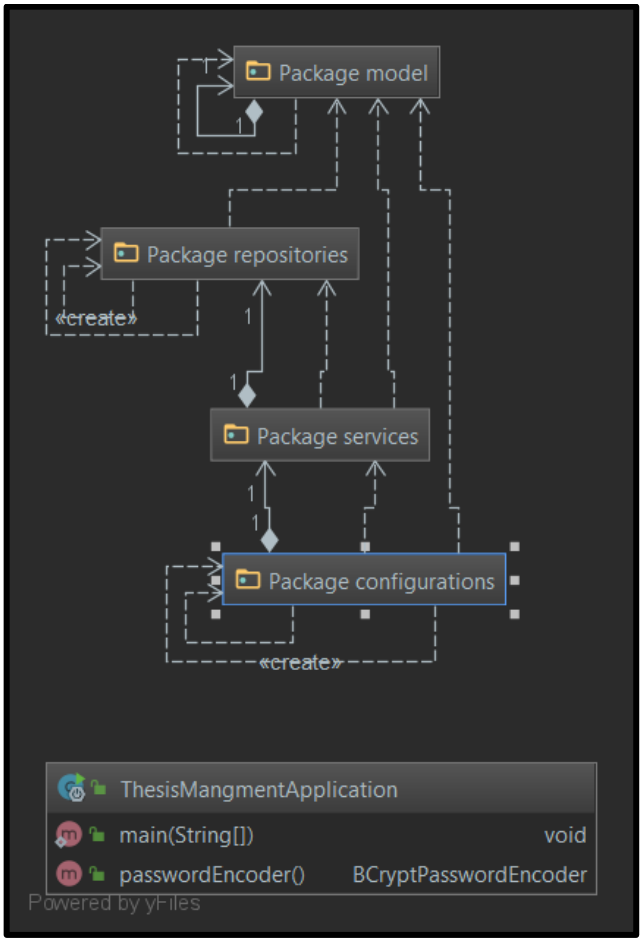
- Coordinator module diagram: This diagram represents the coordinator module. Its structure resembles the structure of previously illustrated modules; however, the implementation varies.



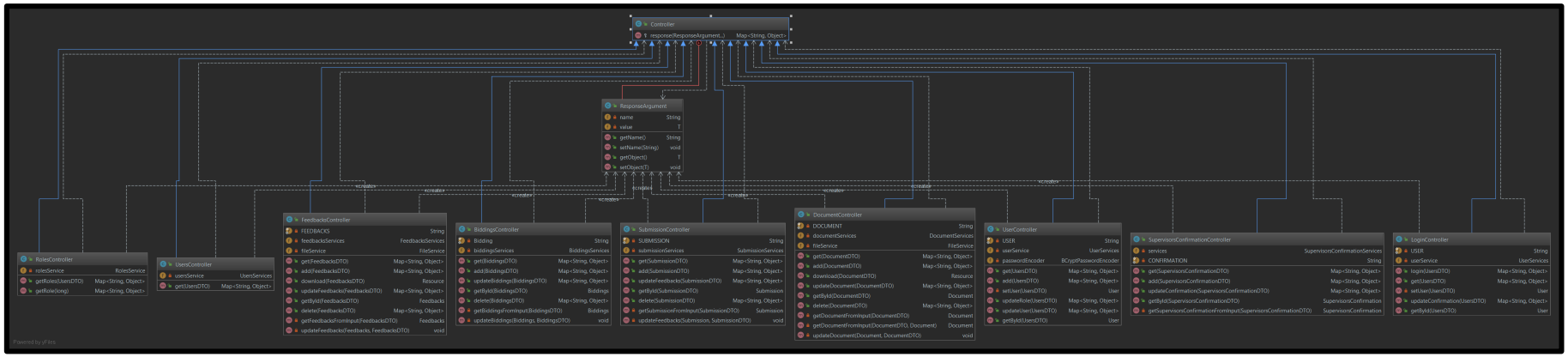
4.2.2 – Back-end Modules

Each of the previously mentioned components in the back-end side of the application has a UML class diagram. The diagrams are presented in order to illustrate how the design principles are, in fact, delivered. First, a package diagram shows the relationship between the modules in the back end. Moreover, classes diagrams show the existing classes, their methods and attributes, and the package that each class belong to. It is important to notice that the classes existing in a package does not have direct relationship to the classes in the other packages.

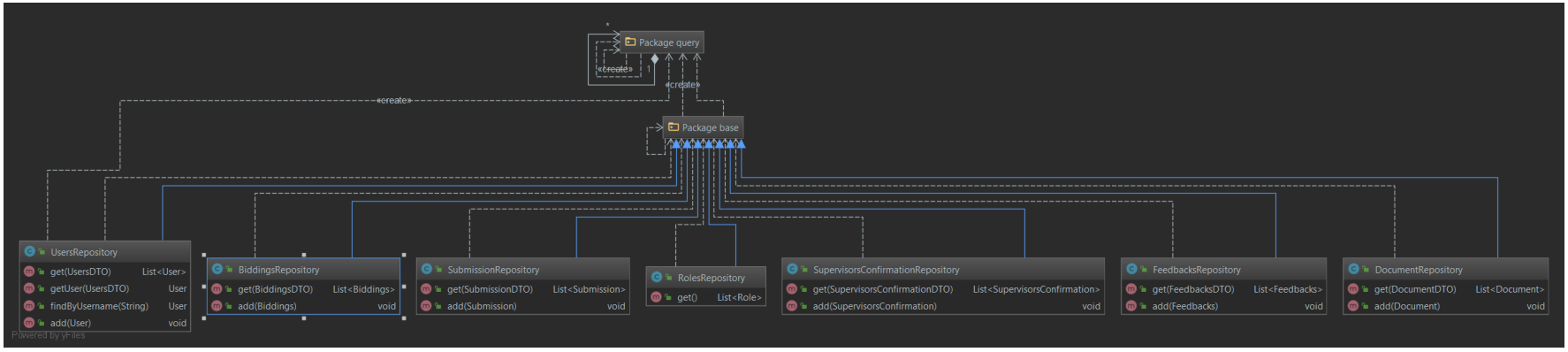
- UML package diagram:



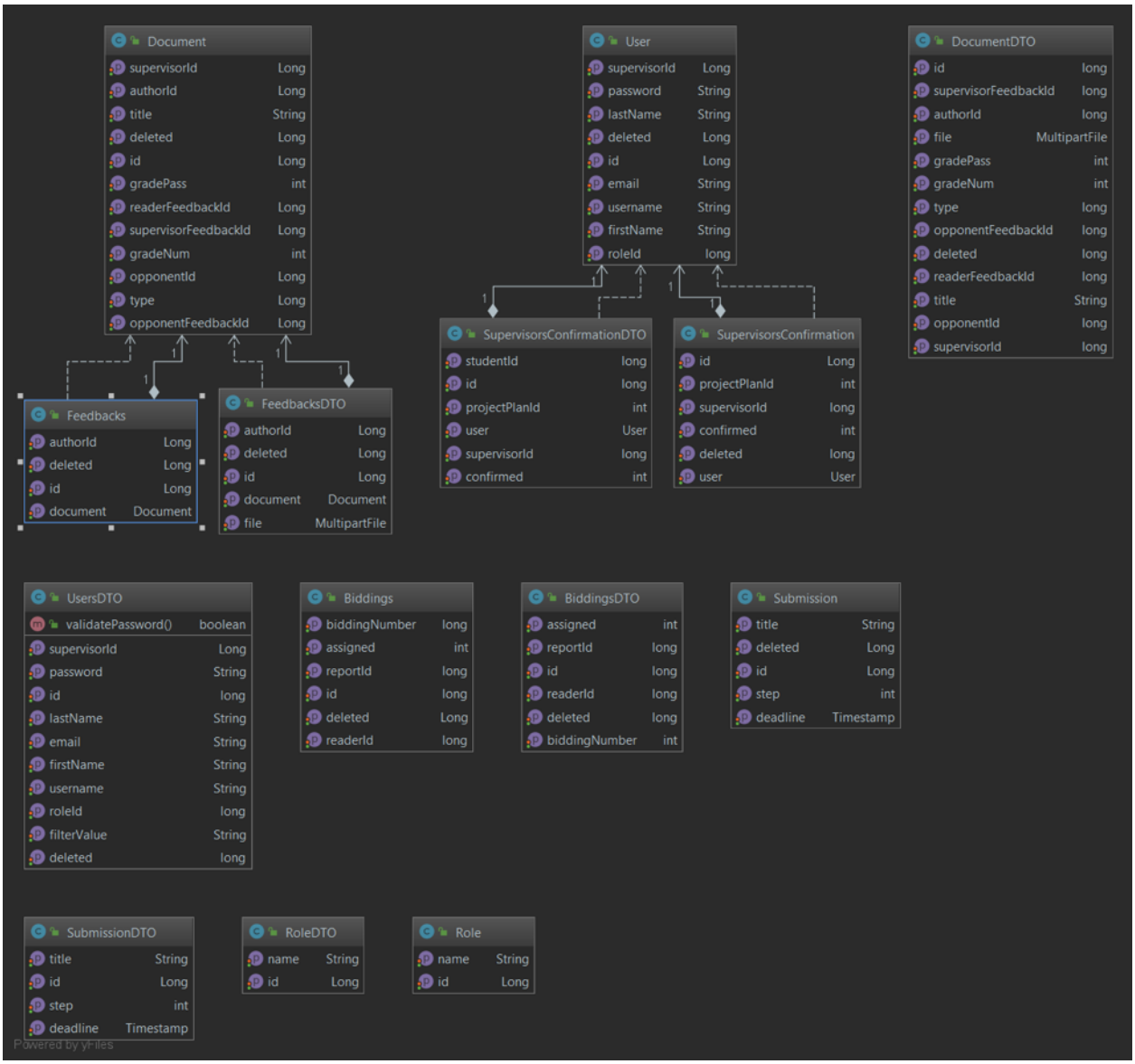
- Controllers module: The class diagram bellow shows mainly the associations between the main controller class and the other sub controller classes such as role, user, feedback, bidding, submission, document, user, and login.



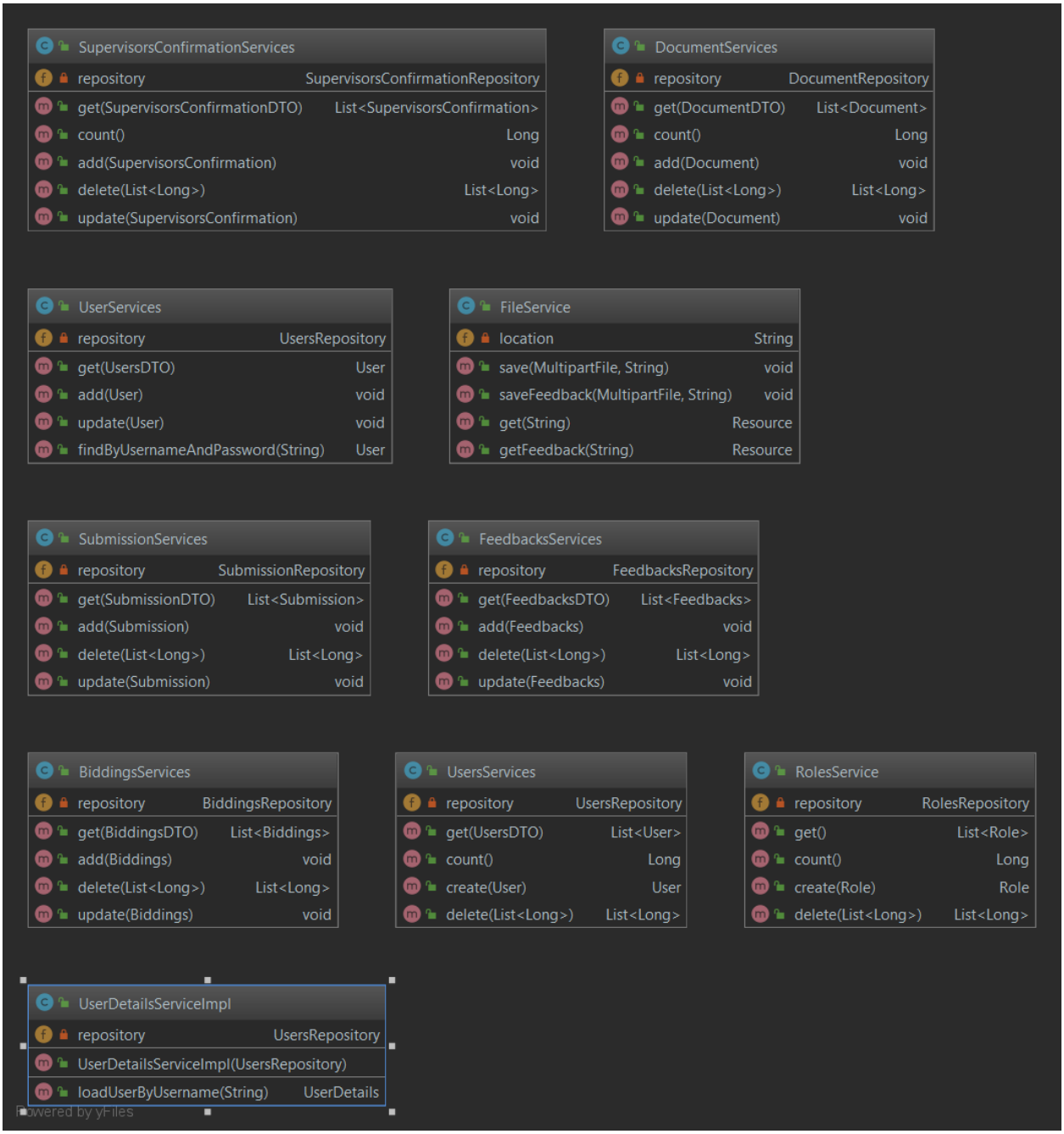
- Repositories module: The class diagram bellow shows mainly the associations between the main base repository class and the other repositories classes such as role, user, feedback, bidding, submission, document, user, and login. Also, it shows the relation between the base repository package and the query package.



- Models module: The class diagram bellow shows the following classes role, user, feedback, bidding, submission, document, user, and login.



- Services module: The class diagram bellow shows the services of the following classes role, user, feedback, bidding, submission, document, user, and login.



- Configuration package: The class diagram bellow shows the classes which are responsible for the authentication of logging in, and encrypt the password in the database.