Thesis Manager

Design Document

Author: Robin Larsson,
Adell Tatrous,
Angelo Spadea,
Helena Tevar,
Jiahui Le,
Yinlong Yao,
Meysam M. Fard

Semester: Spring 2019
Subject: Software Engineering - Design
Course code: 2DV603

Index

Introduction

The design document objective is to describe the design for the Thesis Management system, how it will be constructed and ensure that the requirements will be reached. In order to reach this objective, the design document will provide an explanation of the architecture and design.

Purpose

The design document should be able to be understood by Team E and used as an aid during the process of development. This document contains a high-level description of the design and architecture of the system needed to reach the requirements presented below.
This system is intended to be able to manage the computer science faculty thesis from the applicant students by the allowed staff. Users would be able to log in and out of the system (F-U1/U2). Students will be able to submit different types of assignments and check feedback and grades from the staff related users (student category requirements).
During the process of testing and developing, group E will include in the system the possibility of creating assignments, giving feedback, grading assignments, assigning different roles and functionalities to the users of the system (coordinator, reader,
opponent category requirements).

Definitions, acronyms and abbreviations

- LNU: Linnaeus University.
- CS: Computer Science.

For further definitions, please refer to the Requirement Document.

References

Thesis Manager Requirement Document.
This document will be mostly used in concordance with the requirement document for this same project.

General priorities

The design process follows the guideline created by a group of priorities defined below:

Modularity	As a good practice to improve the team productivity and code clarity, we will put effort into achieving modularity, understood as the property of our system to be loosely coupled and cohesive.
Usability	As the requirements ask us to create a system that can be used successfully, the team has to provide a system that can be used by our consumers with efficiency and satisfaction.
Stability	Thesis Manager has to be reliable when deployed, ensuring that components as the database servers or the web application are uptime as much as possible.
Ease of Implementation	The system should be implemented with ease between the team by encouraging communication and clean code.

Outline of the design

The system will follow a combination of the layered architecture comprised of persistence, service and presentation layers and an MVC architecture for the presentation layer. The data is stored in the database and accessed by the persistence layer. The service layer will manage all the business logic operations and communicate with the persistence layer to access and modify data. Finally, the presentation layer will display the data in the browser and receives requests from users for data manipulation. It will call services provided by the service layer for this purpose.

More information about the architecture and design of the system is provided in their respective sections in this document.

Major design issues

During the process of design, the team encountered a set of issues that were solved as shown below:

Deadline management

Issue 1	Where should we handle the deadlines of assignments?
Option 1.1	Create a `SubmissionEvent-class` that contains information like the type of submission that is open, start date, and end date.
Option 1.2	Create a `Deadline-class` that holds all deadlines.
Decision	The team decided to follow Option 1.2 because the maintainability of the code and possible refactorizations will be applied with less effort.

Software architecture

Issue 2	Which software architecture design should we implement on the server side?
Option 2.1	Model-view-controller
Option 2.2	Monolithic application
Decision	The team decided to follow the model-view-controller architectural pattern in order to provide high cohesion and low coupling for our system.

User multi-role

Issue 3	How do we make sure that a single User can have many different roles at the same time?
Option 3.1	Create a Set in the User class that contains different Roles. The role is an Enum.
Option 3.2	Create a class for each role that each extends a User class.
Decision	Option 3.1 was chosen since it allows us to easily allow a User to have any number of roles. It is also easy to add or remove roles.

Adding new users

Issue 4	How do we add new users to the system?
Option 4.1	Include an Admin role that has the capability to add new users.
Option 4.2	Add each user manually in the database.
Decision	Adding new users manually into the database would have a high cost on maintenance and it would be time-consuming. By having a role as a system administrator, we can create new users and store them in the database. The team decided to apply option 4.1.

Role assignment

Issue 5	How do we assign roles to users already in the system?
Option 5.1	Create functionality for Admins to assign roles to users.
Option 5.2	Add roles to users manually in the database.
Decision	Automation of functionalities will ease the implementation and maintenance of the system. The team decided to use option 5.1.

Coordinator and Supervisor Services

Issue 6	How do we deal with the coordinator and supervisor service?
Option 6.1	Add coordinator and supervisor service in the StudentService-class.
Option 6.2	Create two classes for coordinator and supervisor service called `CoordinatorService-class` and `SupervisorService-class` separately.
Decision	Coordinator and supervisor are in close contact with the student service, however, the team decided to go for two different services that will handle their specific functionalities as said in option 6.2.

Issue 7	How do we deal with the kind of Document in class diagram?
Option 7.1	Create each type Document class
Option 7.2	Create a enumeration class for each type of Document
Decision	Option 7.2. Enums provide compile-time type safety and prevent from comparing constants in different enums.

Issue 8	How do we deal with other services except student service and admin service?
Option 8.1	Create separate class for each service and in each class, implement their own functionalities.
Option 8.2	Create separate class for each service and an abstract class to hold similar functionality of each service. In each class, implement their own specialized functionalities.
Decision	Other services have same or similar behavior, so it is suitable to implement an abstract class as said in option 8.2.

Issue 9	How do we deal with classes in the Repository package?

Option 9.1	Create separate class for each repository.
Option 9.2	Create an interface in the Repository package and each class implement functionalities from the interface.
Decision	The main functionality in Repository class is CRUD, so we decide Option 9.2.

Issue 10	How do we deal with calling each service?
Option 10.1	Using Controller package and Repository package control each service
Option 10.2	Only using Repository call the service
Decision	Option 10.2. In our current design, Controller package is not necessary.

Design details

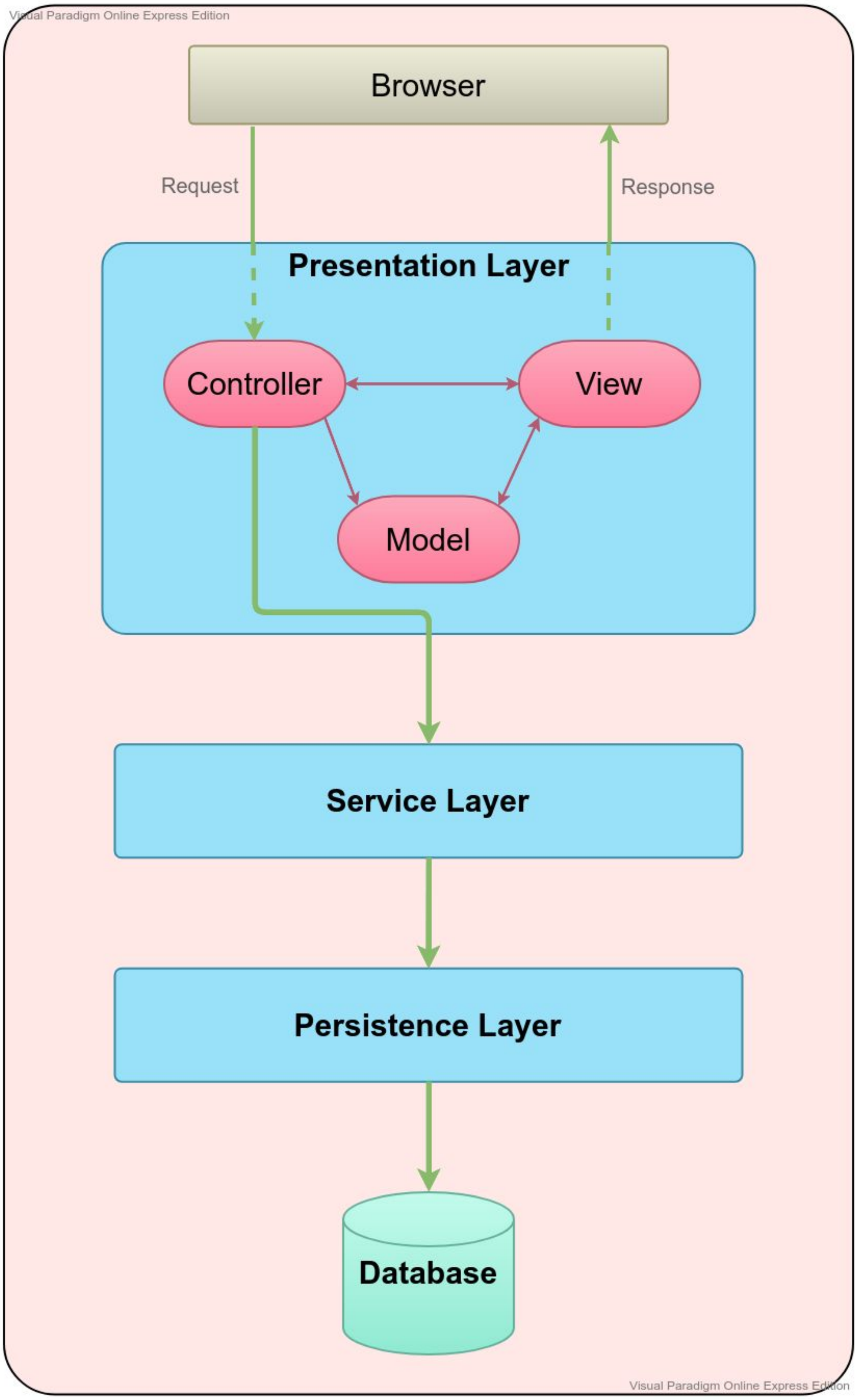
Software Architecture

As presented in the following diagram, this web application utilizes a combination of two architectural patterns, layered architecture and model-view-controller (MVC). The layered architecture comprises three main layers. As the application data is stored in a database, a persistence layer communicates with the database in order to retrieve data from the database and get data persisted in it by means of CRUD operations. It is also responsible for the conversion of the model data objects into database entities and vice-versa. On top of the persistence layer, a service layer exists which implements the business logic of the system. When a data persistence or retrieval is needed in a business operation, the service layer will delegate that to the persistence layer. Lastly, a Presentation layer is responsible for receiving user's Http request from the browser and providing the requested view. Each layer is only conscious about the layer directly below it and can receive services using interfaces.

Additionally, the presentation layer implements an MVC pattern. The model is the representation of the data. A data retrieval or modification request is received from the browser as an Http request which is handled by a suitable controller. The controller, in turn, calls upon services provided by the service layer to update the model data and point out to the correct view. The suitable view is then rendered and presented to the user.

By utilizing a layered architecture, separation of concerns is achieved where different layers can be developed and modified independently when the interfaces are specified. Also, any addition or modification to an interface of a layer will affect only the layer adjacent to it from above. The separation of concerns and keeping related components together increase the cohesion and reduce the coupling, and hence contribute to software maintainability and its quality.

The MVC architecture will also help the development process by separating the concerns and software maintainability. It also lets the development team to be divided into back-end and front-end teams which creates a favorable environment for the developers and will make the process easier to implement.



Architecture

Set of Components

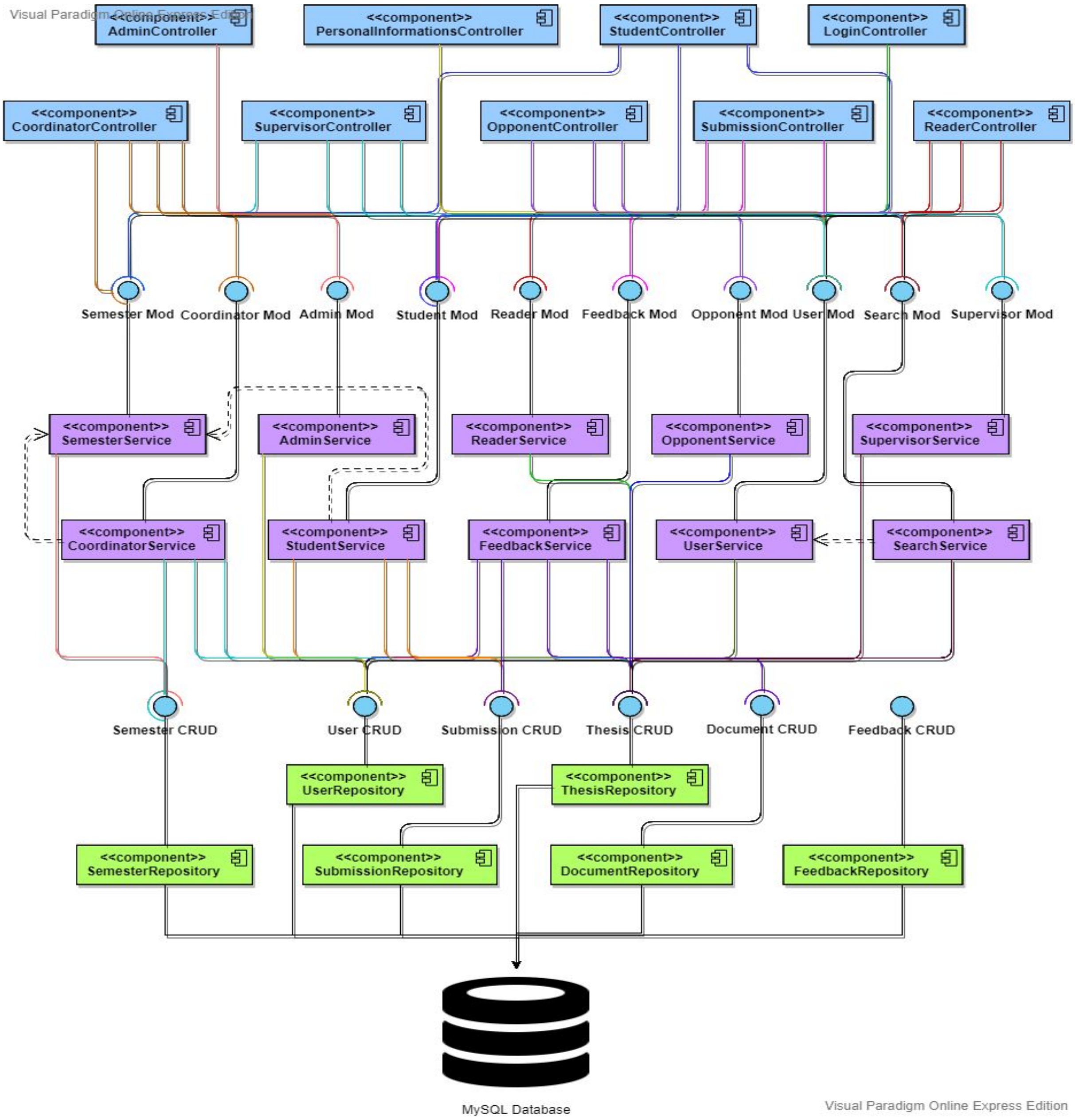
The Thesis Manager system consists of four layers of components; nine controllers, ten services, six repositories and the database; in this order.
Every two components, in adjacent layers or in the same layer only, that need to communicate with each other can do so by a dedicated interface. However, if theses two components are not from the same layer, only the component in the upper layer of the architecture can start the communication with the one below it. So, the system consists of a total of sixteen interfaces; ten interfaces provided by the services and six interfaces provided by the repositories.

Below is a table describing every interface in the system:
Note:
All `Mod` interfaces apply different functionalities and modifications on different objects and convey them to/from their repositories.
All `CRUD` interfaces apply CRUD functionalities on objects. Additionally, they receive search requests with specific parameters and filters and reply with the found objects.

Name	**Provider Component**	**Requirer Components**	**Targeted Objects**
Semester Mod	SemesterService	StudentController, CoordinatorController and SupervisorController	Semester
Coordinator Mod	CoordinatorService	CoordinatorController	Thesis, user and semester
Admin Mod	AdminService	AdminController	User
Student Mod	StudentService	StudentController and SubmissionController	Thesis, user and submission
Reader Mod	ReaderService	ReaderController	Thesis
Feedback Mod	FeedbackService	SubmissionController	Thesis, user, document
Opponent Mod	OpponentService	OpponentController	Thesis
User Mod	UserService	PersonalInformationsController, StudentController, LoginController, CoordinatorController, SupervisorController, OpponentController, SubmissionController and ReaderController	User

Search Mod	SearchService	CoordinatorController, SupervisorController, OpponentController and ReaderController	User and thesis
SupervisorMod	SupervisorService	SupervisorController	Thesis
Semester CRUD	SemesterRepository	SemesterService and CoordinatorService	Semester
User CRUD	UserRepository	AdminService, CoordinatorService, StudentService, FeedbackService and UserService	User
Submission CRUD	SubmissionRepository	StudentService and FeedbackService	Submission
Thesis CRUD	ThesisRepository	ReaderService, OpponentService, SupervisorService, CoordinatorService, StudentService, FeedbackService and SearchService	Thesis
Document CRUD	DocumentRepository	FeedbackService	Document
Feedback CRUD	FeedbackRepository	N/A	Feedback

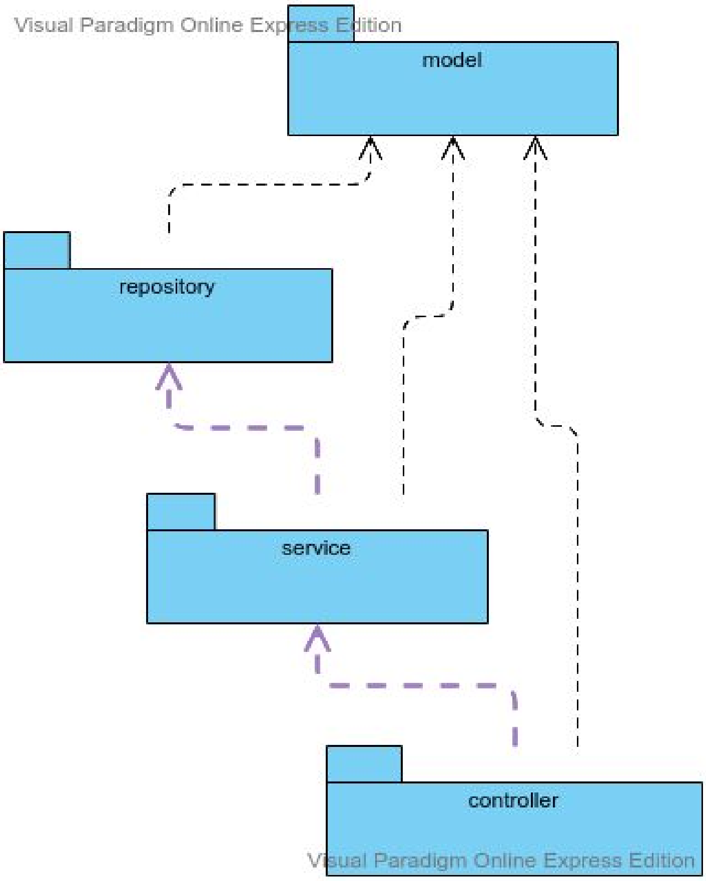
UML Component Diagram


Visual Paradigm Online Express Edition
<<component>> AdminController
<<component>> PersonalInformationsController
<<component>> StudentController
<<component>> LoginController
<<component>> CoordinatorController
<<component>> SupervisorController
<<component>> OpponentController
<<component>> SubmissionController
<<component>> ReaderController
Semester Mod
Coordinator Mod
Admin Mod
Student Mod
Reader Mod
Feedback Mod
Opponent Mod
User Mod
Search Mod
Supervisor Mod
<<component>> SemesterService
<<component>> AdminService
<<component>> ReaderService
<<component>> OpponentService
<<component>> SupervisorService
<<component>> CoordinatorService
<<component>> StudentService
<<component>> FeedbackService
<<component>> UserService
<<component>> SearchService
Semester CRUD
User CRUD
Submission CRUD
Thesis CRUD
Document CRUD
Feedback CRUD
<<component>> UserRepository
<<component>> ThesisRepository
<<component>> SemesterRepository
<<component>> SubmissionRepository
<<component>> DocumentRepository
<<component>> FeedbackRepository
MySQL Database
Visual Paradigm Online Express Edition

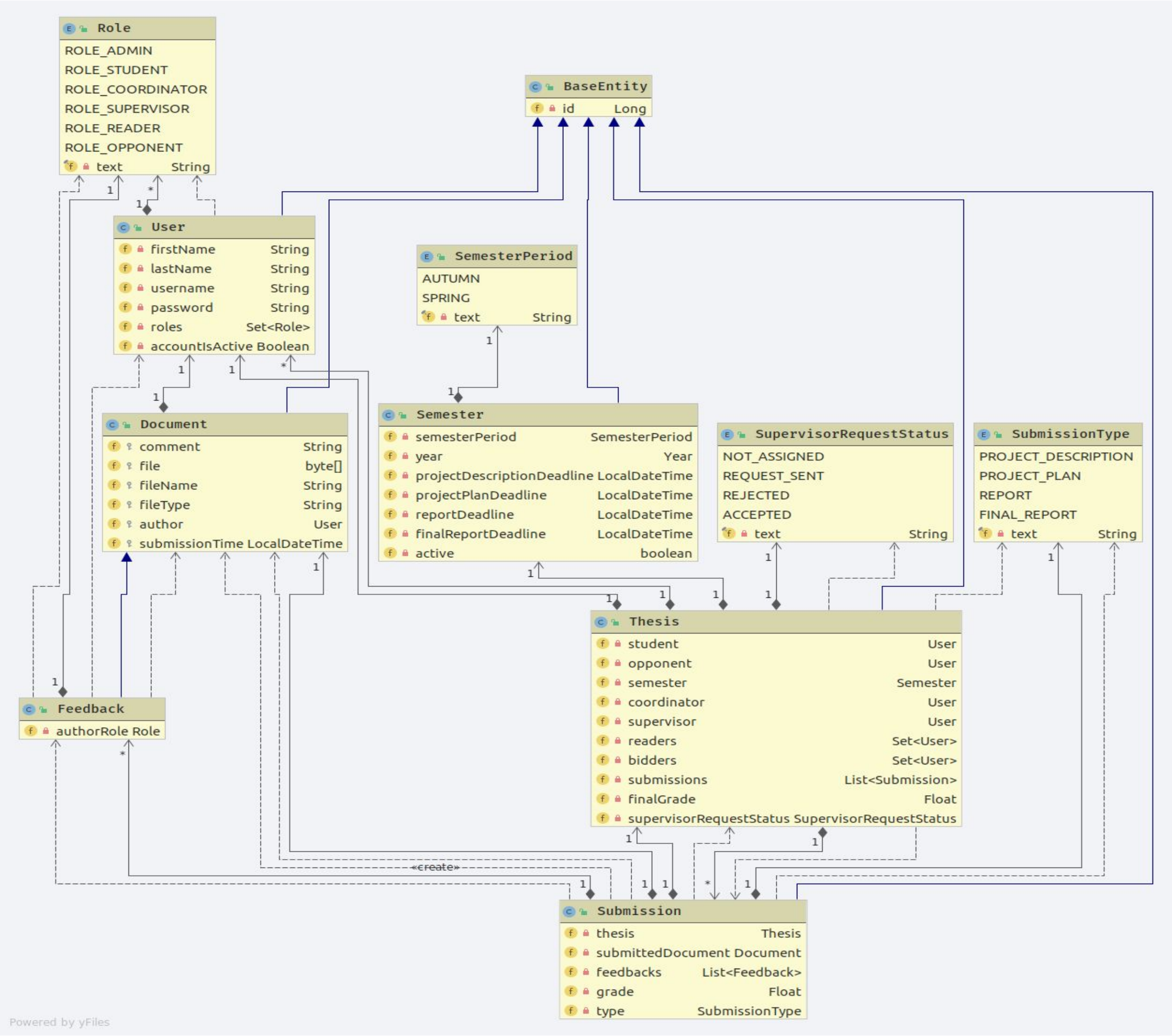
Component Implementation

UML Class Diagram

The application consists of four main packages in order to reflect the architectural choice: model, repository, service and controller. As observable in the following diagram, any package has access to model. However, persistence related operations are done in repository package, data manipulations and business logic are done in service package and transferring data to and from the view is done in controller package.


Visual Paradigm Online Express Edition
model
repository
service
controller
Visual Paradigm Online Express Edition

`Model` package encompasses classes that represent data model that is used throughout the system. The classes in this package are presented in the following class diagram. They are Java EE entities and their specific Spring implementation. The *User* class represents any type of user whose roles are specified in a list of objects of enumeration class *Role*. Documents are represented using either a *Document* object or a *Feedback* object depending on their nature. The main class for storing data is *Thesis* class. One Thesis object is dedicated for each student which stores every related entity, such as student submitted documents, received feedbacks and people with different roles that are associated with that thesis.



Class diagram of package `Model`

Classes in `repository` package extend the JpaRepository of Spring framework which perform CRUD operations on the model classes. They are responsible for the conversion of model data to and from database objects and providing CRUD operations on them.

Business operations will be provided by the `service` package. There is one service class for each actor in the system that provides the functionalities required by that actor. Additionally, other services are provided for additional needed operations. These classes would call classes in repository package to retrieve their required data in order to perform the requested operations. Subsequently, they will call repository classes after any data manipulation in order to have any modifications on data persisted.

The purpose of the `controller` package and its classes is routing and providing the appropriate view (html and its required data) to be rendered in the browser. Additionally, they handle any request made by users. Once they receive a request for a page or data manipulation, they call upon the respective class in service package to fetch the required data for the requested page or get the requested operation done.

Test Plan

In this section, we will describe the process of testing the system, how would be implemented and executed following the test cases described in the *Requirement Document* provided. In order to consider this project finished, the team wishes for the system to be evaluated and prove the fulfillment of the requirements.

Scope

The testing process occur in different areas. The static part of the system is tested to evaluate the documentation reports in order to search for possible errors or ambiguities and perform a refinement of them. The system is also analysed with dynamic tests, that include manual and automatic testing. Manual testing will provide information about the usability of the system once deployed, the result will prove that the system is able to be used for our customers. Automated testing will analyze the correctness of the code in both levels, manual and automated. Manual tests ensure the usability on the system provided in the test cases, and automated testing will evaluate the correctness of the test code.

Testing levels

Manual Testing	
Functional Testing	The team executes each test case with valid data to analyse that the system responds with the valid results or the expected errors or warnings with it is supplied with incorrect data. The test cases can be found in the *Test Cases Document.*
Acceptance Testing	The customer, or the team in case the customer is not available, executes a set of tests that follow the requirements and analyse the results in order to evaluate the project as ready for delivery. This testing process would be performed ad hoc for this system and may be supported by usability testing in case it is needed.
Alpha Quality Testing	This testing process is done in last place when all testing evaluations are positive, all issues are addressed and the product is ready for deployment. The team will evaluate the whole project and provide documentation to prove correctness and completeness.
Automated Testing	This type of testing would be performed with white box testing by using JUnit jupiter api and Mockito framework.
Unit Testing	The system is developed with a set of JUnit test suite executed to test the correctness of each class.
Integration Testing	The developers will add integration testing for each module developed that will test the integration and aggregation of the system.

Traceability Matrix

Req. Sets	Req.	Req.	Req.	Req.	Req.	Req.	Req.	Req.	Req.

	Tested	User	Student	Superv.	Coord.	Reader	Oppon.	Admin.	System
Test Cases	123	50	44	13	22	9	4	3	3
TFU1.1 - 15	2	2							
TFU2.1 - 5	1	1							
TST1.1 - 2	4	3	1						
TST1.3	3	1	2						
TST1.5 - 7	6	3	3						
TST2.1	2		2						
TST2.2 - 3	1		1						
TST4.1	2	1	1						
TST4.2	2	1	1						
TST4.3	3	1	2						
TST4.4	4	1	3						
TFST5.1 - 2 TFST5.4 - 5	5	1	4						
TFST5.3	4	1	3						
TFST6.1 - 2	1		1						
TFST7.1 - 2 TFST7.4 - 6	6	1	5						
TFST7.3	5	1	4						
TFST8.1 - 3	1		1						
TFST9.1 - 2 TFST9.4 - 9	7	1	6						
TFST9.3	6	1	5						
TFST10	1		1						
TFST11.1 - 3	1		1						
TFST12.1 - 3	1		1						
TFC1.1 - 2 TFC2.2	1				1				
TFC2.1	1				1				
TFC3	1				1				
TFC4.1 - 2	2				2				
TFC5.1 - 3	3				3				
TFC6	1				1				

TFC7.1 - 2	1		1				
TFC8.1	1		1				
TFC9.1 - 2	2		2				
TFC10.1	1		1				
TFC11.1	2		2				
TFC12.1			2				
TFC13.1	1		1				
TFC14.1	2		2				
TFC15.1	1		1				
TFC16.1	1		1				
TFSU1.1 - 4	1	1					
TFSU2	1	1					
TFSU3.1 - 2	2	2					
TFSU4	1	1					
TFSU5	1	1					
TFSU6.1 - 2	1	1					
TFSU7.1 - 2	1	1					
TFR1.1 - 2	1			1			
TFR2	1			1			
TFR3.1 - 2	1			1			
TFR4	1			1			
TFR5.1 - 2	1			1			
TFR6	1			1			
TFO1.1 - 2	1				1		
TFO2.1 - 2	2				2		
TFA1	1					1	
TFA2.1 - 2	2					2	
TNSY1.1	1						1
TNSY2.1	1						1
TNSY3.1	1						1

Results

Automated testing was performed by using JUnit, Spring boot test and Mockito. Each developer worked concurrently with developing and testing. In order to commit new functionalities, all the code should be tested and those test passed. This type of testing would assure correctness of the code, however extended manual testing was completed to ensure the system reached its goal.

Manual testing followed the implementation of functionalities and was performed by the developers during and after the automated testing (white box) and for a different member of the team (black box). The team decided that the success of 90% of the manual tests is the requirement to consider the project finished.

Test code	Title	Requirements	Status
TFU1.1	Test user login-student is able to login	F-U1, N-U2	PASS
TFU1.2	Test user login-student username not correct	F-U1, N-U2	PASS
TFU1.3	Test user login-student password not correct	F-U1, N-U2	PASS
TFU1.4	Test user login- supervisor is able to login	F-U1, N-U2	PASS
TFU1.5	Test user login- supervisor username not correct	F-U1, N-U2	PASS
TFU1.6	Test user login-supervisor password not correct	F-U1, N-U2	PASS
TFU1.7	Test user login-coordinator is able to login	F-U1, N-U2	PASS
TFU1.8	Test user login-coordinator username not correct	F-U1, N-U2	PASS
TFU1.9	Test user login-coordinator password not correct	F-U1, N-U2	PASS
TFU1.10	Test user login-reader is able to login	F-U1, N-U2	PASS
TFU1.11	Test user reader username not correct	F-U1, N-U2	PASS
TFU1.12	Test user login. Reader password not correct	F-U1, N-U2	PASS
TFU1.13	Test user login-opponent is able to login	F-U1, N-U2	PASS
TFU1.14	Test user reader username not correct	F-U1, N-U2	PASS
TFU1.15	Test user login-opponent password not correct	F-U1, N-U2	PASS
TFU2.1	Test user-student log out	F-U2	PASS
TFU2.2	Test user-supervisor log out	F-U2	PASS
TFU2.3	Test user-coordinator log out	F-U2	PASS
TFU2.4	Test user-reader log out	F-U2	PASS
TFU2.5	Test user-opponent log out	F-U2	PASS

TST1.1	Test project description submission by students	F-ST1, N-U6, N-U7, N-U8	PASS
TST1.2	Test project description submission by students with wrong file	F-ST1, N-U6	PASS
TST1.3	Test project description submission by students after deadline	F-ST1, N-ST2, N-U6	PASS
TST1.5	Test project description submission by students when failed on deadline	F-ST1, F-ST3, N-ST6, N-U6	PASS
TST1.6	Test project description submission by students with wrong file size	F-ST1, N-U6, N-U7	PASS
TST1.7	Test project description submission by students with big comment	F-ST1, N-U6, N-U8	PASS
TST2.1	Test student is able to check project description grade: pass	F-ST2, N-ST4	PASS
TST2.2	Test student checks project description grade with deadline ongoing	F-ST2	PASS
TST2.3	Test student checks project description grade after deadline.	F-ST2	PASS
TST4.1	Test project plan submission by students	F-ST4, N-U6	PASS
TST4.2	Test project plan submission by students with wrong file	F-ST1, N-U6	PASS
TST4.3	Test project plan submission by students after deadline	F-ST4, N-ST2, N-U6	PASS
TST4.4	Test project plan submission by students when file already in system	F-ST1, N-ST3, N-ST6, N-U6	PASS
TFST5.1	Re-submit a project plan	F-ST5, N-U6, N-ST16, N-ST17, N-ST18	PASS
TFST5.2	Re-submit a project plan with the wrong format of submission	F-ST5, N-U6, N-ST16, N-ST17, N-ST18	PASS
TFST5.3	Re-submit a project plan after deadline	F-ST5, N-U6, N-ST17, N-ST18	PASS
TFST5.4	Re-submit a project plan more than once	F-T5, N-U6, N-ST16, N-ST17, N-ST18	PASS
TFST5.5	Re-submit project plans	F-ST5, N-U6, N-ST16, N-ST17, N-ST18	PASS

TFST6.1	Submit a supervisor request	F-ST6	PASS
TFST6.2	Submit a supervisor request when the supervisor is unavailable	F-ST6	PASS
TFST7.1	Submit a report	F-ST7, N-U6, N-ST8, N-ST9, N-ST10, N-ST17	PASS
TFST7.2	Submit a report with wrong format of submission	F-ST7, N-U6, N-ST8, N-ST9, N-ST10, N-ST17	PASS
TFST7.3	Submit a report after deadline	F-ST7, N-U6, N-ST9, N-ST10, N-ST17	PASS
TFST7.4	Submit a report more than once	F-ST7, N-U6, NST-8, N-ST9, N-ST10, N-ST17	PASS
TFST7.5	Submit reports	F-ST7, N-U6, NST-8, N-ST9, N-ST10, N-ST17	PASS
TFST7.6	Submit reports without receiving a passing grade from a supervisor	F-ST7, N-U6, NST-8, N-ST9, N-ST10, N-ST17	PASS
TFST8.1	View one submitted feedback on the project description	F-ST8	PASS
TFST8.2	View more than one submitted feedback on the project description	F-ST8	PASS
TFST8.3	View none submitted feedback on the project description	F-ST8	PASS
TFST9.1	Submit a final report	F-ST9, N-U6, N-ST11, N-ST12, N-ST13, N-ST14, N-ST15	PASS
TFST9.2	Submit a final report with wrong format of submission	F-ST9, N-U6, N-ST11, N-ST12, N-ST13, N-ST14, N-ST15	PASS
TFST9.3	Submit a final report after the deadline	F-ST9, N-U6, N-ST12, N-ST13, N-ST14, N-ST15	PASS
TFST9.4	Submit a final report more than once	F-ST9, N-U6, N-ST11,	PASS

		N-ST12, N-ST13, N-ST14, N-ST15	
TFST9.6	Submit a final report	F-ST9, N-U6, N-ST11, N-ST12, N-ST13, N-ST14, N-ST15	PASS
TFST9.7	Submit final report without receiving feedback from supervisor	F-ST9, N-U6, N-ST11, N-ST12, N-ST13, N-ST14, N-ST15	PASS
TFST9.8	Submit final report without receiving feedback from reader	F-ST9, N-U6, N-ST11, N-ST12, N-ST13, N-ST14, N-ST15	PASS
TFST9.9	Submit report without receiving feedback from opponent	F-ST9, N-U6, N-ST11, N-ST12, N-ST13, N-ST14, N-ST15	PASS
TFST10	View the grade of the final report	F-ST10	PASS
TFST11.1	View one submitted feedback on the project plan	F-ST11	PASS
TFST11.2	View more than one submitted feedback on the project plan	F-ST11	PASS
TFST11.3	View none submitted feedback on the project plan	F-ST11	PASS
TFST12.1	View one submitted feedback on the report	F-ST12	PASS
TFST12.2	View more than one submitted feedback on the report	F-ST12	PASS
TFST12.3	View none submitted feedback on the report	F-ST12	PASS
TFC1.1	Set a deadline for one submission	F-C1	PASS
TFC1.2	Set deadlines for more than one submissions	F-C1	PASS
TFC2.1	Change a deadline for one submission	F-C2	PASS
TFC2.2	Change deadlines for more than one submissions	F-C1	PASS
TFC3	See a list of all submitted project plans	F-C3	PASS
TFC4.1	See all the submitted project plans' status	F-C4, N-C1	PASS
TFC4.2	See all the submitted project plans' status without supervisor's feedback and supervision confirmation	F-C4, N-C1	PASS

TFC5.1	Submit a grade on the project plan	F-C5, N-C2, N-C3	PASS
TFC5.2	Submit a grade on the project plan	F-C5, N-C2, N-C3	PASS
TFC5.3	Submit a grade on the project plan without supervisor’s feedback and supervision confirmation	F-C5, N-C2, N-C3	PASS
TFC6	See a list of readers who have made a bid for a a report	F-C6	PASS
TFC7.1	Test assign a reader to a report	F-C7	PASS
TFC7.2	Test assign readers to a report	F-C7	PASS
TFC8.1	Coordinator checks on opponents list	F-C8	PASS
TFC9.1	Coordinator assigns opponent to a report	F-C9, N-C5	PASS
TFC9.2	Coordinator assigns opponent to a report already assigned	F-C9, N-C5	PASS
TFC10.1	Coordinator checks list of final evaluations	F-C10	PASS
TFC11.1	Coordinator grades final reports	F-C11, N-C6	PASS
TFC12.1	Coordinator grades project description	F-C12, N-C7	PASS
TFC13.1	Coordinator checks a project description	F-C13	PASS
TFC14.1	Coordinator checks a project plan	F-C14, N-C1	PASS
TFC15.1	Coordinator checks a final report	F-C15	PASS
TFC16.1	Coordinator checks list of project descriptions	F-C16	PASS
TFSU1.1	A supervisor views incoming supervision requests from students. The students have submitted a supervision request	F-SU1	PASS
TFSU1.2	A supervisor views incoming supervision request form students after deadline. The students have submitted a supervision requests	F-SU1	FAIL
TFSU2	A supervisor views the project description submitted by a student requesting supervision	F-SU2	PASS
TFSU3.1	A supervisor rejects a supervision request	F-SU3, N-SU2	PASS
TFSU3.2	A supervisor accepts a supervision request	F-SU3, N-SU2	PASS
TFSU4	A supervisor shall be able to give feedback to a	F-SU4	PASS

	project plan		
TFSU5	A supervisor view the evaluation of a project plan by the coordinator	F-SU5	PASS
TFSU6.1	A supervisor views her students' reports	F-SU6	PASS
TFSU7.1	A supervisor shall be able to submit her assessment on a student's report	F-SU7	PASS
TFR1.1	The students have submitted the reports. A reader see a list of reports	F-R1	PASS
TFR1.2	The students have not submitted the reports. A reader see a list of reports.	F-R1	PASS
TFR2	A reader bid for as many reports as desired	F-R2	PASS
TFR3.1	A reader see a list of reports assigned to him/her. After a coordinator has assign reports for a reader.	F-R3	PASS
TFR3.2	A reader see a lists of reports assigned to him/her. A coordinator has no assign reports for a reader.	F-R3	PASS
TFR4	A reader submit feedback on the reports	F-R4	PASS
TFR5.1	The students have submitted the final reports. A reader see a lists of final reports assigned to him/her.	F-R5	PASS
TFR5.2	The students have not submitted the final reports. A reader see a list of final reports assigned to him/her.	F-R5	PASS
TFR6	A reader submit a final evaluation of the final reports	F-R6	PASS
TFO1.1	Coordinator has assigned reports for a reader. An opponent see a list of reports assigned to him/her.	F-O1	PASS
TFO1.2	Coordinator has not assigned reports for a reader. An opponent see a list of reports assigned to him/her.	F-O1	PASS
TFO2.1	An opponent see a list of reports assigned to him/her before report deadline	F-O1, N-O1	PASS
TFO2.2	An opponent see a lists of reports assigned to him/her after report deadline	F-O1, N-O2	PASS
TFA1	Add new users to the system	F-A1	PASS
TFA2.1	Assign roles to users	F-A2, N-A1	PASS

TFA2.2	Assign wrong roles to users	F-A2, N-A1	PASS
TNSY1.1	Check system testability	N-SY1	PASS
TNSY2.1	Check web application	N-SY2	PASS
TNSY3.1	Check application technologies	N-SY3	PASS

Ad-hoc testing

During the development process new functionalities have been implemented in addition to the ones originally created in the Requirements Document, as they are referred in this document as “additional requirements”. In order to test those functionalities, the team created a group of tests ad-hoc for them.

Test code	Title	Status
AH-001	Search Thesis works in Coordinator user.	PASS
AH-002	Search Thesis works in Supervisor user.	PASS
AH-003	Search Thesis works in Reader user.	PASS
AH-004	Search Thesis works in Opponent user.	PASS
AH-005	The grade on the report assessment is a float.	PASS
AH-006	Coordinator sees the status of the projects.	PASS
AH-007	A user shall be able to do all their authorized functionalities from the search results.	PASS

Code review

The system has been going through analysis to monitor the technical quality in relation with architecture and design by using Sonargraph. The team was focused on its objectives as explained on *general priorities*, trying to create code with low coupling and high cohesion. The results of the analysis proved that the intention from the team is reflected in the code. The packages that needed to be more flexible and adaptable got higher values in instability while the packages that are more constrictive got a low value in their instability.

Sonargraph resulted on a value of 0 on abstractness for the whole system, having four modules with abstractness 0. Only repository (module that include only interfaces) and services are the only ones that reached more than 0.1 points on abstractness. None of the packages got in the useless zone of distance and we reached a zero value in most of them, having only the most strict package to reach the value -1, as expected for models, a package with low abstraction, as it was intended. Sonargraph also showed the results of the structure analysis. The system maintainability level reached 91 points over 100 and the structural debt index is 14 points.

The system contains a small cycle concentrated between two classes inside the models package, thesis and submission, two classes that are extremely connected. Thesis is the most important object managed in our system, so the team decided that in order to avoid the use of multiple objects in our system, to focus on thesis and give it more responsibilities. The team considered the cycle enough encapsulated in between this two classes to not be a problem for the architecture of the system.

The system was stress tested in order to get the results required to answer the non-functional requirement that assuming that all the N students will try to submit the report during the last 10 minutes before the deadline, and that it is required a response time for the "Submit report" action below 10 seconds. In order to get the results the team used the software “Postman” to get the following result: run time = one minute ; avg response time = 1841ms ; number of responses = 32. Further calculations showed the total requests the system is able to hold as proved: $Nmax = 32 \times 60 \div 6 = 320$ requests.