Diploma Projects Management App

Sprint Report

<GEORGIOS KARATHANOS>

VERSIONS HISTORY

Date	Version	Description	Author
2023/05/01	1.0	Overall writing of the report after the implementation of the application	Georgios Karathanos

1 Introduction

This document provides information concerning all the project.

1.1 Purpose

The purpose of this web application is to make easier the assignments of thesis topics from professors to students. As in many Universities that happens manually, this implementation suggests a solution to this problem.

1.2 Document Structure

The rest of this document is structured as follows. Section 2 describes our Scrum team and specifies the Sprint's backlog. Section 3 specifies the main design concepts for this release of the project.

2 Scrum team and Sprint Backlog

2.1 Scrum team

Product Owner	Georgios Karathanos
Scrum Master	Georgios Karathanos
Development Team	Georgios Karathanos (solo project)

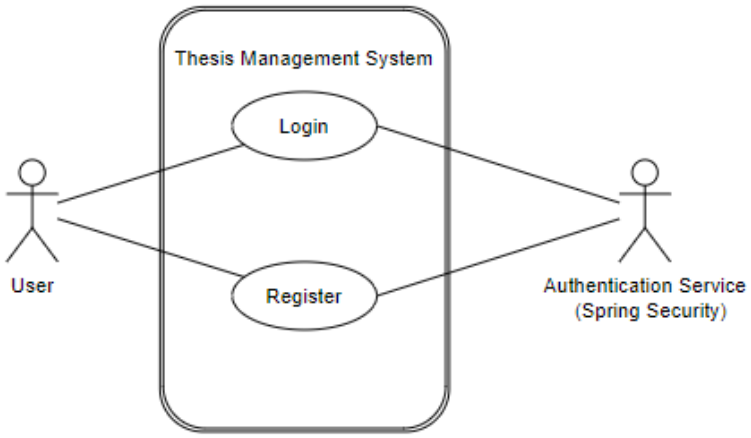
2.2 Sprints

Sprint No	Begin Date	End Date	Number of weeks	User stories
1	**2023/11/04**	**2023/16/04**	**1**	**U1, U2, S1, P1, P2, P3, P4**
2	**2023/17/04**	**2023/22/04**	**1**	**S2, S3, S4, P5, P6, P7, P8, P9**
3	**2023/23/04**	**2023/28/04**	**1**	**Testing**

3 Use Cases

We will present the use cases of the system grouped by the type of user they serve (general user, student, professor). First, we will show the use case diagram and then the descriptions.

General User



Picture 1: User's use case diagram.

3.1 <Use Case 1 - Register>

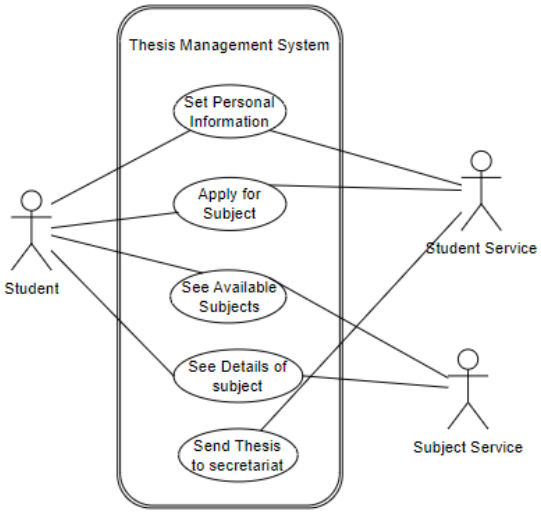
Use case ID	1
Actors	General User
Preconditions	-
Main flow of events	1. The use case starts when the user clicks the button 'Register' on the main page. 2. The user is redirected to a registration page. 3. The user fills in the information that is required for the registration and clicks 'Register'. 4. The user is redirected to the login page and is informed about his successful registration with a message.

Alternative flow 1	Username already exists. The user is informed in that case with a message.
Post conditions	The user has an account and can use the application.

3.2 <Use Case 2 - Login>

Use case ID	2
Actors	General User
Preconditions	Have an account
Main flow of events	1. The use case starts when the user clicks the button 'Login' on the main page. 2. The user is redirected to a login page. 3. The user fills in his username and password and clicks 'Login'. 4. The user is redirected to his dashboard.
Alternative flow 1	Username and password don't make a valid user match. The user is informed in that case with a message and the login does not happen.
Post conditions	The user can use the provided functionalities of the application depending on his role.

Student



Picture 2: Student's use case diagram.

3.3 <Use Case 3 - Set Personal Information for Student>

Use case ID	3
Actors	Student
Preconditions	Be logged in to the application
Main flow of events	1. The use case starts when the student clicks the button 'Manage Personal Information' on his dashboard. 2. The student is redirected to a page that has a form with his current information. 3. The student fills in every field he wants to update and clicks 'Save'. 4. The student is redirected to his dashboard.
Alternative flow 1	The fields Year, Grade and Remaining courses have some restrictions about the values that they can take. For example, the user can't submit the changes until he enters nonnegative number of remaining courses.
Post conditions	The student's information is updated.

3.4 <Use Case 4 – See Available Subjects>

Use case ID	4
Actors	Student
Preconditions	Be logged in to the application
Main flow of events	1. The use case starts when the student clicks the button 'View Available Subjects' on his dashboard. 2. The student is redirected to a page that has a table with every subject that is available. I) If there are no available subjects the user is informed about that at the beginning of the table
Post conditions	The student sees all the available subjects listed.

3.5 <Use Case 5 - See Details of a Subject>

Use case ID	5
Actors	Student
Preconditions	Be logged in to the application
Main flow of events	1. The use case starts when the student clicks the button 'View Available Subjects' on his dashboard. 2. The student is redirected to a page that has a table with every subject that is available. 3. For each subject there is a button 'View more info' and the user clicks on it to continue. 4. The student is redirected to a page that has all the details of the subject including the supervisor's information.
Post conditions	The student sees the description of the subject that he is interested.

3.6 <Use Case 6 – Apply for Subject>

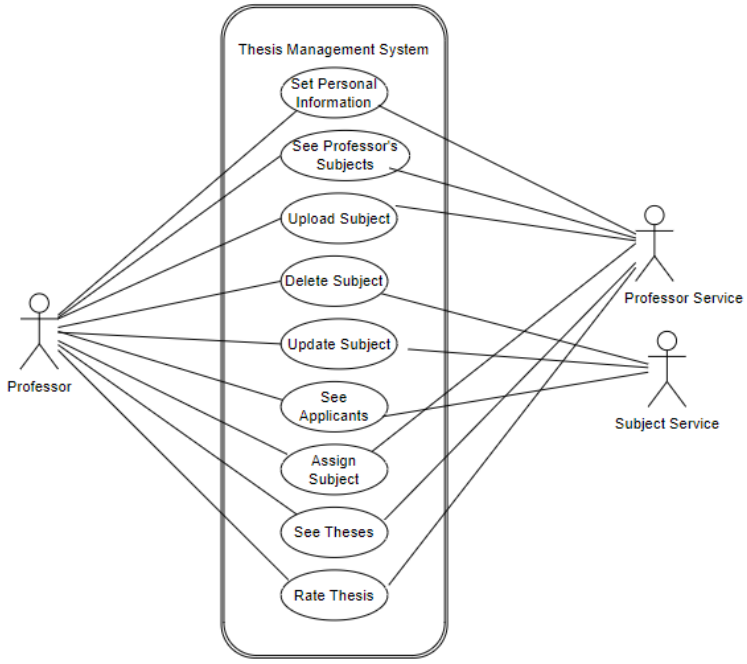
Use case ID	6
Actors	Student
Preconditions	Be logged in to the application
Main flow of events	1. The use case starts when the student clicks the button 'View Available Subjects' on his dashboard. 2. The student is redirected to a page that has a table with every subject that is available. 3. For each subject there is a button 'View more info' and the user clicks on it to continue. 4. The student is redirected to a page that has all the details of the subject including the supervisor's information. 5. The student writes a message in a text field that is available for the professor explaining why he is interested and then clicks on 'Apply for this subject'.

Alternative flow 1	The student already has a supervisor and is waiting for the results. In that case he is already bound with a thesis and can't apply for another subject. The system informs him about that when he is gets on the page with the subject's description.
Alternative flow 2	The student is a graduate. In that case he can't apply ever again.
Post conditions	A new application has been created.

3.7 <Use Case 7 – Send thesis results to the secretariat>

Use case ID	7
Actors	Student
Preconditions	Be logged in to the application / Have an evaluated thesis
Main flow of events	1. The use case starts when the student clicks the button 'Manage your thesis' on his dashboard. 2. The student is redirected to a page that shows him the final grade of his thesis. 3. The student clicks on the 'Send this to the Secretariat of the School'. 4. The student is redirected to a new page where he is informed that his thesis was evaluated successfully.
Alternative flow 1	The thesis has not been evaluated yet. In that case the page instead of showing the mark informs the student about that.
Alternative flow 2	The student hasn't found a supervisor yet. In that case the page instead of showing the mark informs the student about that.
Alternative flow 3	The student is a graduate. In that case the page instead of showing the mark informs the student about that.
Post conditions	The thesis is deleted, and the student becomes graduate. He is no longer able to apply for any subject.

Professor



Picture 3: Professor's use case diagram.

3.8 <Use Case 8 - Set Personal Information for Professor>

Use case ID	8
Actors	Professor
Preconditions	Be logged in to the application
Main flow of events	1. The use case starts when the professor clicks the button 'Manage Personal Information' on his dashboard. 2. The professor is redirected to a page that has a form with his current information. 3. The professor fills in every field he wants to update and clicks 'Save'. 4. The professor is redirected to his dashboard.
Post conditions	The professor's information is updated.

3.9 <Use Case 9 – See Professor's Subjects>

Use case ID	9
Actors	Professor
Preconditions	Be logged in to the application
Main flow of events	1. The use case starts when the professor clicks the button 'Manage your subjects' on his dashboard. 2. The professor is redirected to a page that has a table with all the subjects that he has uploaded.
Post conditions	The professor sees all his subjects listed.

3.10 <Use Case 10 – Upload Subject>

Use case ID	10
Actors	Professor
Preconditions	Be logged in to the application
Main flow of events	1. The use case starts when the professor clicks the button 'Manage your subjects' on his dashboard. 2. The professor is redirected to a page that has a table with all the subjects that he has uploaded. 3. The professor clicks the button '+Add new thesis subject' that is on the top of the page 4. The professor gets redirected to a page that has a form with the necessary information he needs to fill in order to upload a new subject. 5. The professor fills in the form and clicks on 'Save Subject'.
Post conditions	A new subject is uploaded.

3.11 <Use Case 11 – Delete Professor's Subject>

Use case ID	11
Actors	Professor
Preconditions	Be logged in to the application / Have subjects of his own
Main flow of events	1. The use case starts when the professor clicks the button 'Manage your subjects' on his dashboard. 2. The professor is redirected to a page that has a table with all the subjects that he has uploaded. For each subject there is a 'Delete' button. 3. The professor clicks the 'Delete' button of the subject that he wants to delete 4. The system asks if he is sure, and the professor can cancel or verify the deletion.
Post conditions	The subject gets deleted.

3.12 <Use Case 12 – Update Professor's Subject>

Use case ID	12
Actors	Professor
Preconditions	Be logged in to the application / Have subjects of his own
Main flow of events	1. The use case starts when the professor clicks the button 'Manage your subjects' on his dashboard. 2. The professor is redirected to a page that has a table with all the subjects that he has uploaded. For each subject there is an 'Update' button. 3. The professor clicks the 'Update' button of the subject that he wants to delete 4. The professor gets redirected to a page with a form same as the one that he filled when he uploaded the subject. 5. The professor changes every field that he wants and clicks the 'Save Subject' button.
Post conditions	The subject gets updated.

3.13 <Use Case 13 – See Applicants>

Use case ID	13
Actors	Professor
Preconditions	Be logged in to the application / Have subjects of his own
Main flow of events	1. The use case starts when the professor clicks the button 'Manage your subjects' on his dashboard. 2. The professor is redirected to a page that has a table with all the subjects that he has uploaded. For each subject there is a 'View Applicants' button. 3. The professor clicks the 'View Applicants' button of the subject that he wants to navigate. 4. The professor gets redirected to a page where is a table that has all the applicants listed.
Post conditions	The professor sees the applicants of a specific subject.

3.14 <Use Case 14 – Assign Subject>

Use case ID	14
Actors	Professor
Preconditions	Be logged in to the application / Have subjects of his own
Main flow of events	1. The use case starts when the professor clicks the button 'Manage your subjects' on his dashboard. 2. The professor is redirected to a page that has a table with all the subjects that he has uploaded. For each subject there is a 'View Applicants' button. 3. The professor clicks the button 'View Applicants' button of the subject that he wants to navigate. 4. The professor gets redirected to a page where is a table that has all the applicants listed. 5. Under the table there is a drop-down menu from which the professor can choose the strategy that the system should follow for the assignment for the thesis. The user chooses a strategy and clicks 'Assign Subject'
Alternative flow 1	There are no applicants. The system redirects the professor to a page that informs him that the assignment was not successful.

Alternative flow 2	There is no candidate that fulfills the restrictions of the assignment strategy. The system redirects the professor to a page that informs him that the assignment was not successful.
Post conditions	The subject is assigned to a candidate. Therefore, a new thesis is created.

3.15 <Use Case 15 – See Theses>

Use case ID	15
Actors	Professor
Preconditions	Be logged in to the application
Main flow of events	1. The use case starts when the professor clicks the button 'Manage your theses' on his dashboard. 2. The professor is redirected to a page that has a table with all the theses that he supervises. For each thesis, in the last column: I. There is a 'Rate' button if the professor has not evaluated it yet. II. There is the final grade of the thesis if the professor has evaluated this thesis and the student has not sent it to the secretariat yet.
Post conditions	The professor sees his theses.

3.16 <Use Case 16 – Rate Theses>

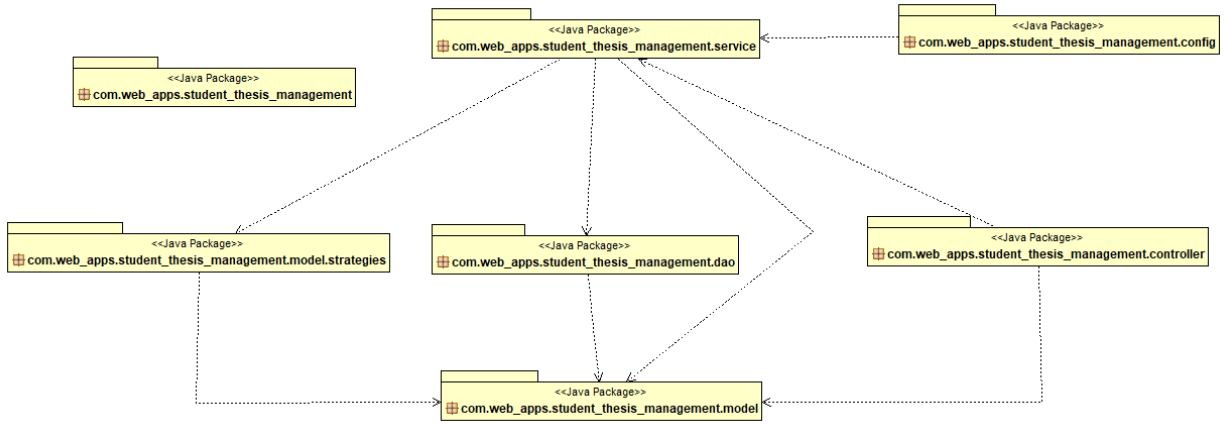
Use case ID	16
Actors	Professor
Preconditions	Be logged in to the application / Be a supervisor of at least one thesis
Main flow of events	1. The use case starts when the professor clicks the button 'Manage your theses' on his dashboard. 2. The professor is redirected to a page that has a table with all the theses that he supervises. For each thesis, in the last column: I. There is a 'Rate' button if the professor has not evaluated it yet. II. There is the final grade of the thesis if the professor has evaluated this thesis and the student has not sent it to the secretariat yet.

	3. The professor clicks the 'Rate' button of the non-evaluated thesis that he wants to evaluate. 4. The professor gets redirected to a page that has a form with the grades that he must fill for the evaluation of the thesis (implementation grade, report grade and presentation grade).
Post conditions	The thesis is rated by the professor.

4 Design

4.1 Architecture

Below there is the package diagram of the backend part of the application.

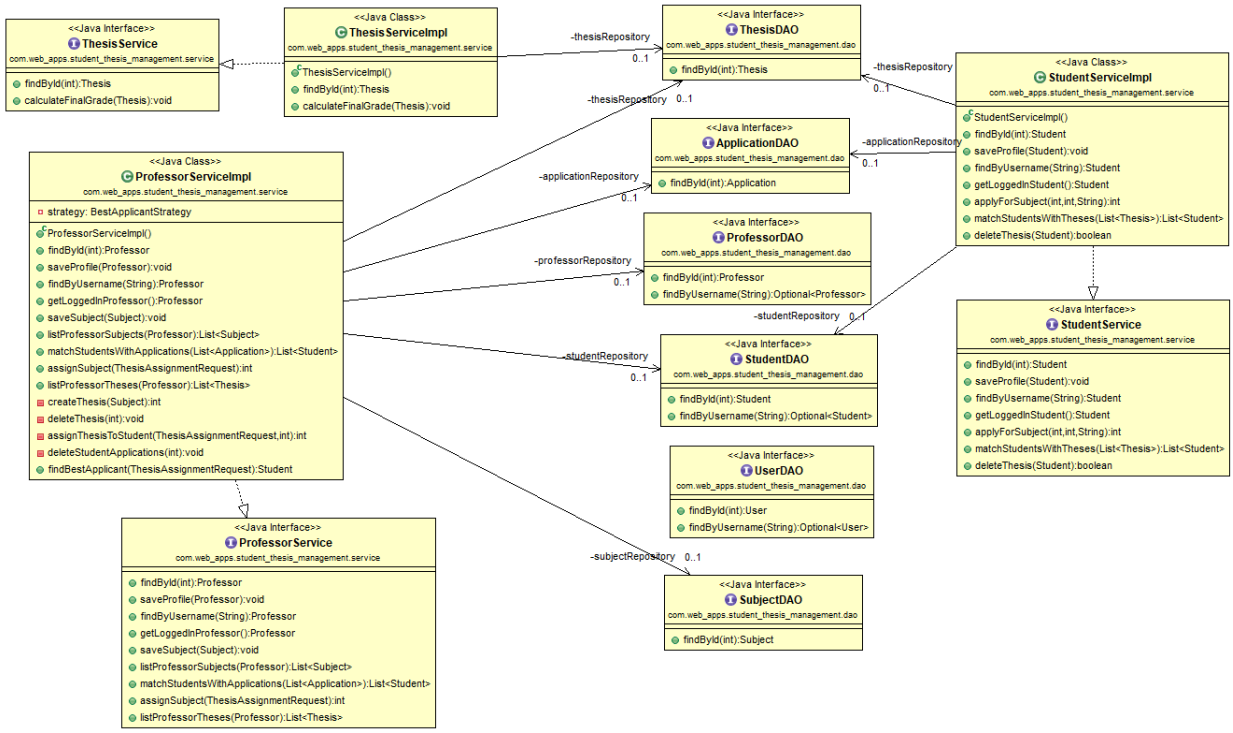


Picture 4: Package diagram.

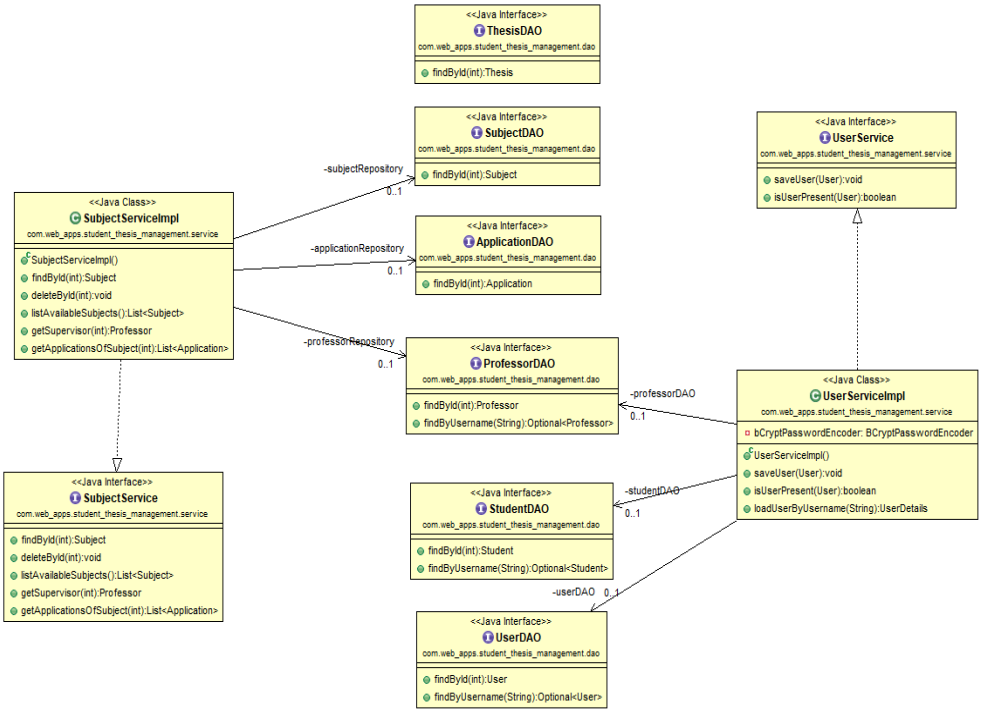
4.2 Design

In this section we will present the class diagrams of the application. We made some groups of classes, so the dependencies are more obvious to the reader. We will take advantage of the MVC architecture that the application is based on, so we will see the dependencies that exist between pairs of layers.

4.2.1 DAO LAYER WITH SERVICE LAYER

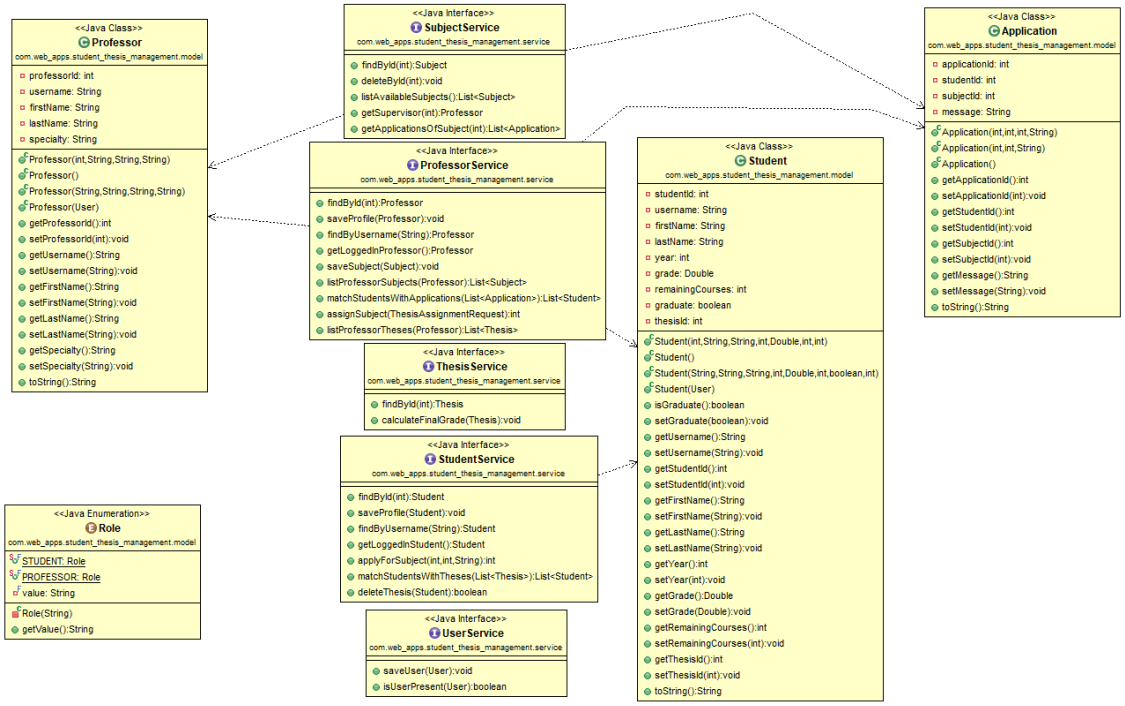


Picture 5: DAO Layer with Service Layer (1).

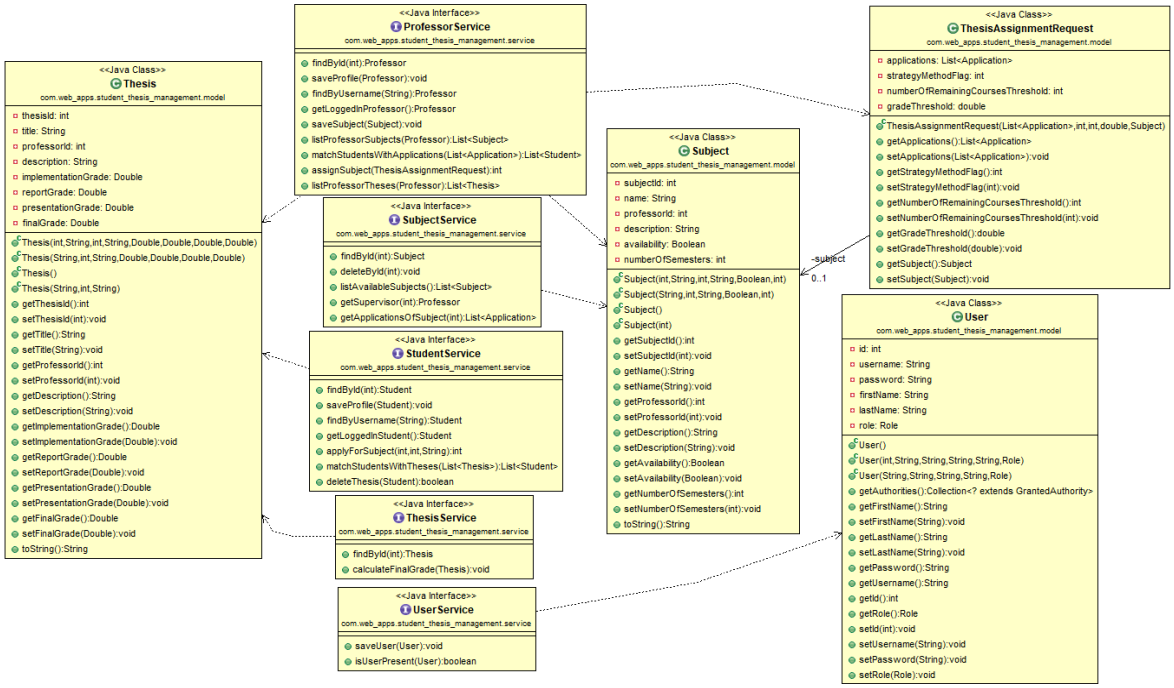


Picture 6: DAO Layer with Service Layer (2).

4.2.2 MODEL LAYER WITH SERVICE LAYER

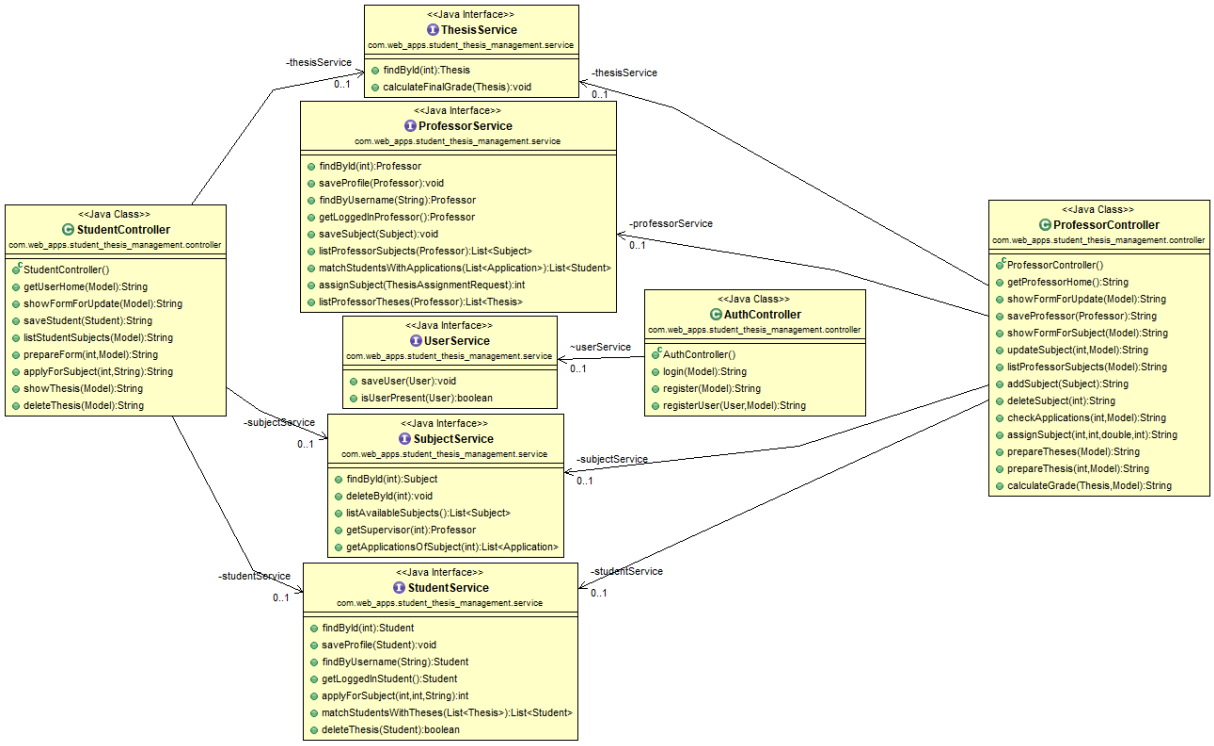


Picture 7: Model Layer with Service Layer (1).



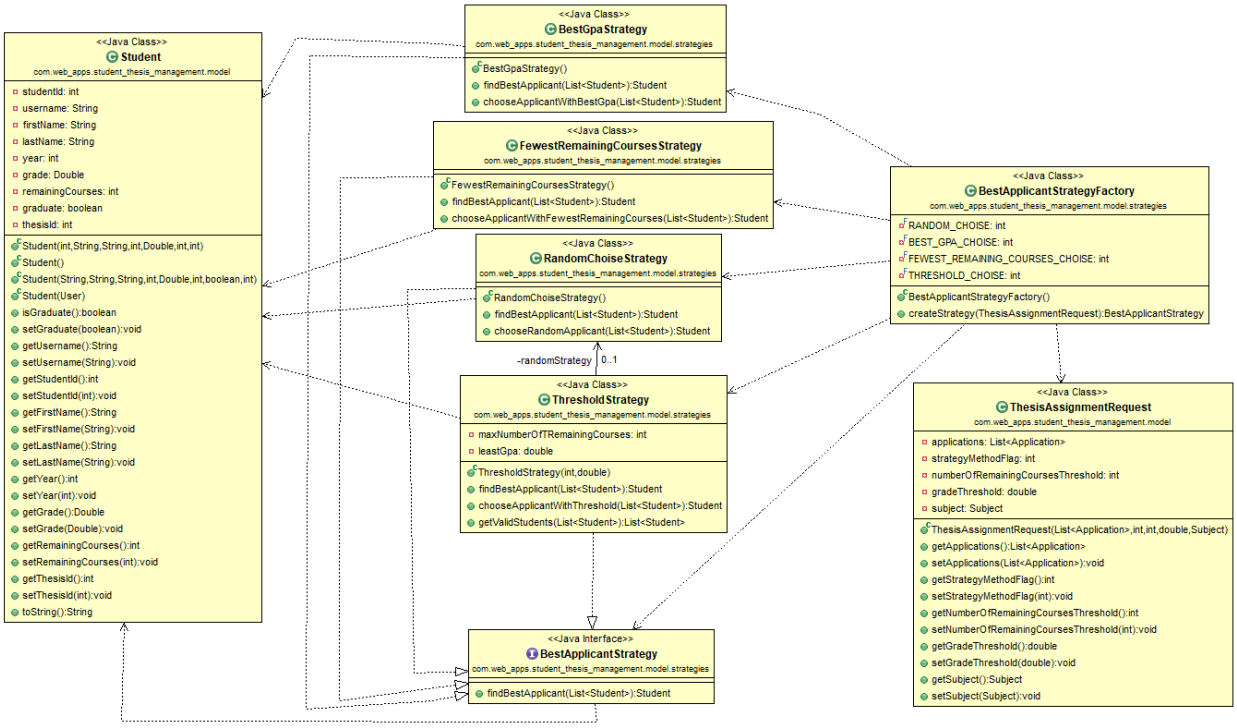
Picture 8: Model Layer with Service Layer (2).

4.2.3 CONTROLLER LAYER WITH SERVICE LAYER



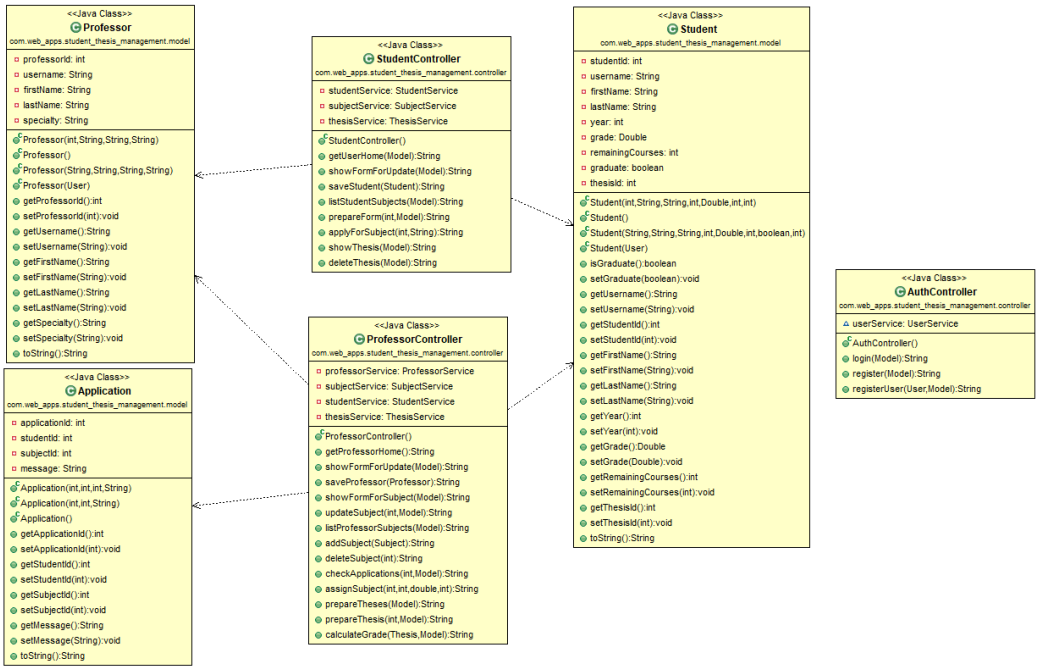
Picture 9: Controller Layer with Service Layer.

4.2.4 MODEL LAYER WITH STRATEGIES

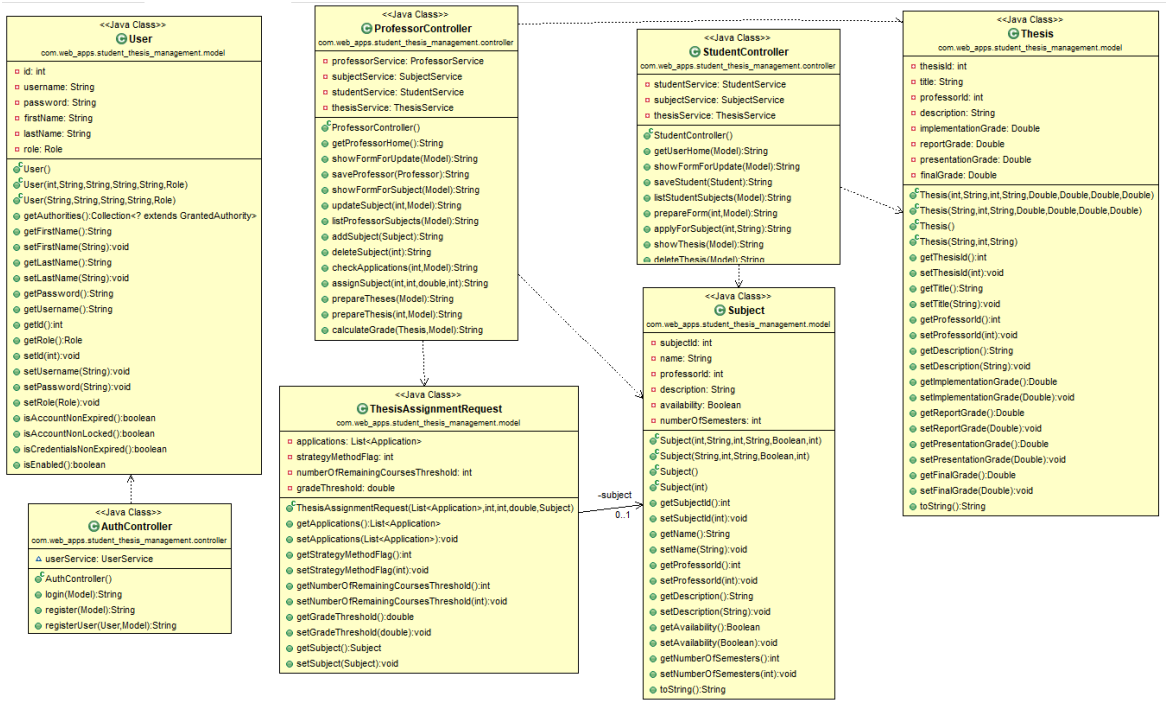


Picture 10: Model Layer and Strategies.

4.2.5 MODEL LAYER WITH CONTROLLER LAYER

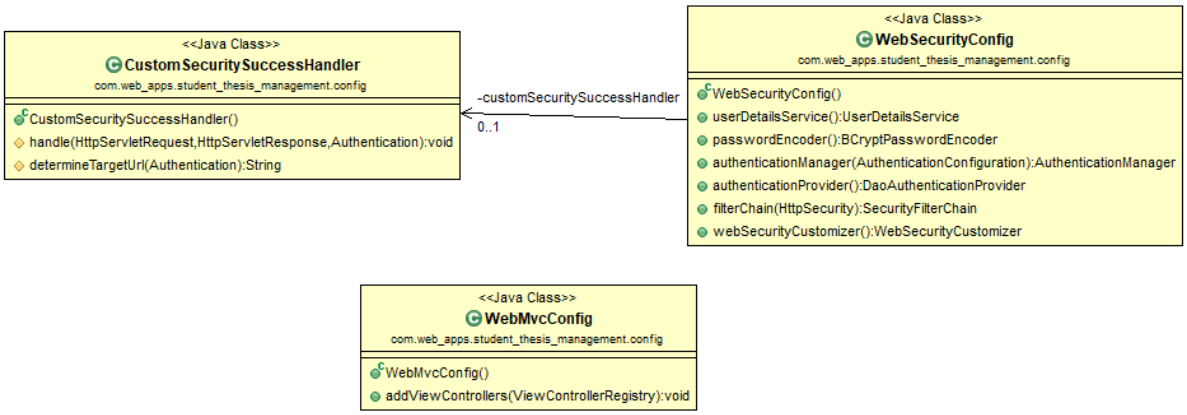


Picture 11: Model Layer with Controller Layer (1).



Picture 12: Model Layer with Controller Layer (2).

4.2.6 CONFIGURATION PACKAGE



Picture 13: Classes that implement the Spring Security dependencies.

4.2.7 CRC CARDS

Class Name: StudentThesisManagementApplication	
Responsibilities: ▪ Contains the 'main' that we need to execute, in order to run the application.	**Collaborations:** -

Class Name: CustomSecuritySuccessHandler	
Responsibilities: ▪ Handles the login request of the user and redirects him to the correct dashboard depending on his role.	**Collaborations:** -

Class Name: WebMvcConfig	
Responsibilities: ▪ Redirects the user to the homepage view when he first attempts to use the application.	**Collaborations:** -

Class Name: WebSecutiryConfig	
Responsibilities: ▪ Is responsible for the authentication setup. ▪ Authorizes each entity of the project.	**Collaborations:** ▪ CustomSecuritySuccsessHandler: sets an object of this class as a success handler.

Class Name: AuthController	
Responsibilities: ▪ Is responsible for handling the HTTP requests that a general user can make via the pages that are under the auth folder	**Collaborations:** ▪ User: prepare user objects for registration ▪ UserService: needs this class to handle the registration of a user.

Class Name: ProfessorController	
Responsibilities: ▪ Is responsible for handling the HTTP requests that a professor can make via the pages that are under the professor folder	**Collaborations:** ▪ Application: uses it to retrieve the applications of a subject and to assign the thesis based on them. ▪ Professor: uses object of this class to represent the logged in controller ▪ Student: uses objects of this class to retrieve the applicants and the students that the professor supervises

	▪ Subject: uses objects of this class to represent the subjects that the professor uploads-deletes-updates and needs to handle the applications. ▪ Thesis: uses objects of this class to retrieve the theses that a professor supervises and to do the evaluation. ▪ ThesisAssignmentRequest: uses this class to zip the information that is necessary for the algorithm that chooses the selected candidate. ▪ ProfessorService: uses it to call the methods that implement the use cases of the professor. ▪ StudentService: uses it to retrieve the students given a list of applications. ▪ SubjectService: uses it to manage subject objects and to retrieve the applications of a specific subject. ▪ ThesisService: uses it to retrieve thesis objects and to evaluate them

Class Name: StudentController	
Responsibilities: ▪ Is responsible for handling the HTTP requests that a student can make via the pages that are under the student folder	**Collaborations:** ▪ Student: uses object of the class to represent the logged in user. ▪ Subject: uses it to represent all the available subjects and the subject that he wants to inspect. ▪ Thesis: uses it to represent the thesis that is assigned to the logged in student. ▪ StudentService: uses it to save the updates of a student, retrieve the logged in student, implement the subject application and delete the thesis when it is sent to the secretariat.

	▪ SubjectService: uses it to retrieve the available subjects and the supervisor of a specific one. ▪ ThesisService: uses it to retrieve the thesis of the logged in student.

Class Name: ApplicationDAO	
Responsibilities: ▪ Is responsible to retrieve an application entry that is saved in the database given its primary key.	**Collaborations:** ▪ Application: An object of this class will represent the database entry that is retrieved.

Class Name: ProfessorDAO	
Responsibilities: ▪ Is responsible to retrieve a professor entry that is saved in the database given its primary key or username.	**Collaborations:** ▪ Professor: An object of this class will represent the database entry that is retrieved.

Class Name: StudentDAO	
Responsibilities: ▪ Is responsible to retrieve a student entry that is saved in the database given its primary key or username.	**Collaborations:** ▪ Student: An object of this class will represent the database entry that is retrieved.

Class Name: SubjectDAO	
Responsibilities: ▪ Is responsible to retrieve a subject entry that is saved in the database given its primary key.	**Collaborations:** ▪ Subject: An object of this class will represent the database entry that is retrieved.

Class Name: ThesisDAO	
Responsibilities: ▪ Is responsible to retrieve a thesis entry that is saved in the database given its primary key.	**Collaborations:** ▪ Thesis: An object of this class will represent the database entry that is retrieved.

Class Name: UserDAO	
Responsibilities: ▪ Is responsible to retrieve a user entry that is saved in the database given its primary key or username.	**Collaborations:** ▪ User: An object of this class will represent the database entry that is retrieved.

Class Name: Application	
Responsibilities: ▪ This class represents the applications in the memory. It has the accessor and getter methods for each field.	**Collaborations:** -

Class Name: Professor	
Responsibilities: ▪ This class represents the professors in the memory. It has the accessor and getter methods for each field.	**Collaborations:** -

Class Name: Role	
Responsibilities: ▪ This Enum has the two possible values that can be the role of a user (‘Student’ or ‘Professor’).	**Collaborations:** -

Class Name: Student	
Responsibilities: ▪ This class represents the students in the memory. It has the accessor and getter methods for each field.	**Collaborations:** -

Class Name: Subject	
Responsibilities: ▪ This class represents the subjects in the memory. It has the accessor and getter methods for each field.	**Collaborations:** -

Class Name: Thesis	
Responsibilities: ▪ This class represents the theses in the memory. It has the accessor and getter methods for each field.	**Collaborations:** -

Class Name: ThesisAssignmentRequest	
Responsibilities:	**Collaborations:** ▪ Application: Have a list of the applications that exist for a specific subject.

▪ This class has object that have as fields all the necessary information for the strategy to choose the applicant that the thesis will be assigned to. It has the accessor and getter methods for each field.	▪ Subject: An object of this class represents the subject that the request is about.

Class Name: User	
Responsibilities: ▪ This class represents the users in the memory and maps these objects with the necessary authentication services. It has the accessor and getter methods for each field.	**Collaborations:** -

INTERFACE CARD

Class Name: BestApplicantStrategy	
Responsibilities: ▪ This interface has the basic method that we want to implement with different algorithm. The method is the one that finds the best applicant.	**Collaborations:** ▪ Student: The applicants are objects of this class.

Class Name: BestApplicantStrategyFactory	
Responsibilities: ▪ This class creates a new object that will implement the strategy. The creation of the object is based on the request that professor sent.	**Collaborations:** ▪ ThesisAssignmentRequest: uses object of this class to represent the request. ▪ RandomChoiseStrategy: creates object of this class if the proportionate strategy was selected based on the request.

	▪ BestGpaStrategy: creates object of this class if the proportionate strategy was selected based on the request. ▪ FewestRemainingCoursesStrategy: creates object of this class if the proportionate strategy was selected based on the request. ▪ ThresholdStrategy: creates object of this class if the proportionate strategy was selected based on the request.

Class Name: BestGpaStrategy	
Responsibilities: ▪ This class implements the strategy with an algorithm that choses the student with the highest GPA.	**Collaborations:** ▪ Student: The applicants are objects of this class.

Class Name: FewestRemainingCoursesStrategy	
Responsibilities: ▪ This class implements the strategy with an algorithm that choses the student with the lowest number of remaining courses.	**Collaborations:** ▪ Student: The applicants are objects of this class.

Class Name: RandomChoiseStrategy	
Responsibilities: ▪ This class implements the strategy with an algorithm that choses the student randomly.	**Collaborations:** ▪ Student: The applicants are objects of this class.

Class Name: ThresholdStrategy	
Responsibilities: ▪ This class implements the strategy with an algorithm that choses the student randomly, but first filters the applicants with some restrictions on GPA and number of remaining courses.	**Collaborations:** ▪ Student: The applicants are objects of this class.

Class Name: ProfessorServiceImpl	
Responsibilities: ▪ This class implements the ProffesorService interface. In other words, this class implements the functionalities that are provided to professors by the application.	**Collaborations:** ▪ Appliation, ApplicationDAO: uses these classes to delete all the application of the accepted student. This happens to prevent confusion with another professor as only one professor can be the supervisor of a student. ▪ Professor, ProfessorDAO: need these classes to either retrieve a professor given an attribute either get the current logged in professor. ▪ Student, StudentDAO: uses these classes to map the application of a subject with the students that made them and also to implement the assignment of the thesis to a specific student. ▪ Subject, SubjectDAO: uses these classes to retrieve all the subjects that a specific professor published, to implement the publishment (meaning the transactions in the database that are necessary) and use subject object for the assignment.

	▪ Thesis, ThesisDAO: needs these classes to create a new thesis and save it to the database, and also to retrieve all the theses that a given professor supervises. ▪ ThesisAssignmentRequest: ▪ BestApplicantStrategy: ▪ BestApplicantStrategyFactory:

Class Name: StudentServiceImpl	
Responsibilities: ▪ This class implements the StudentService interface. In other words, this class implements the functionalities that are provided to students by the application.	**Collaborations:** ▪ Student, StudentDAO: uses these classes to implement any transaction with the database that is necessary and is about student objects. ▪ Thesis, ThesisDAO: needs these classes to delete a student's thesis after it is sent to the secretariat. ▪ Appliation, ApplicationDAO: the logged in user creates and application for a specific subject and saved it into the database.

Class Name: SubjectServiceImpl	
Responsibilities: ▪ This class implements the SubjectService interface. It is responsible to find available subjects and get the supervisor (or the applications) of a specific one.	**Collaborations:** ▪ Professor, ProfessorDAO: get the supervisor of a given subject. ▪ Subject, SubjectDAO: use these classes to retrieve the available subjects from the database, retrieve a specific subject, delete a specific subject and find a specific subject (based on the primary key). ▪ Appliation, ApplicationDAO: get the applications of a given subject.

Class Name: ThesisServiceImpl	
Responsibilities: ▪ This class implements the ThesisService interface. It is responsible to find theses and calculate the final grade of a specific one, given the evaluation of the professor.	**Collaborations:** ▪ Thesis, ThesisDAO: Retrieve a specific thesis or evaluate one and then save it.

Class Name: UserServiceImpl	
Responsibilities: ▪ This class implements the UserService interface. It is responsible to implement the registration of a user and check if a user is already registered.	**Collaborations:** ▪ Professor, ProfessorDAO: Inserts new professor entry in the database in case this role was chosen by the user. ▪ Student, StudentDAO: Inserts new student entry in the database in case this role was chosen by the user. ▪ User, UserDAO: Inserts new user entry in the database using these classes.

Class Name: TestAuthController	
Responsibilities: ▪ This class tests every method of the class AuthController. It simulates HTTP requests and check if the parameters and the results is the expected.	**Collaborations:** ▪ AuthController: This is the class that is tested. ▪ Role: Use this Enum to initialize a user object.

Class Name: TestProfessorController	
Responsibilities: ▪ This class tests every method of the class ProfessorController. It simulates HTTP requests and check if the parameters and the results is the expected.	**Collaborations:** ▪ ProfessorController: This is the class that is tested. All the following classes are used to manage fake-temporary objects that we make in order to implement the tests. ▪ Application, ApplicationDAO ▪ Professor, ProfessorDAO ▪ Student, StudentDAO ▪ Subject, SubjectDAO ▪ Thesis, ThesisDAO ▪ ProfessorService

Class Name: TestStudentController	
Responsibilities: ▪ This class tests every method of the class StudentController. It simulates HTTP requests and check if the parameters and the results is the expected.	**Collaborations:** ▪ StudentController: This is the class that is tested. All the following classes are used to manage fake-temporary objects that we make in order to implement the tests. ▪ Application, ApplicationDAO ▪ Student, StudentDAO ▪ Thesis, ThesisDAO ▪ StudentService

Class Name: TestApplicationDAOJpa	
Responsibilities: ▪ This class tests the ApplicationDAO class with jUnit.	**Collaborations:** ▪ ApplicationDAO: This is the class that is tested. ▪ Application: used to manage fake-temporary object that we make in order to implement the tests.

Class Name: TestProfessorDAOJpa	
Responsibilities: ▪ This class tests the ProfessorDAO class with jUnit.	**Collaborations:** ▪ ProfessorDAO: This is the class that is tested. ▪ Professor: used to manage fake-temporary object that we make in order to implement the tests.

Class Name: TestStudentDAOJpa	
Responsibilities: ▪ This class tests the StudentDAO class with jUnit.	**Collaborations:** ▪ StudnetDAO: This is the class that is tested. ▪ Student: used to manage fake-temporary object that we make in order to implement the tests.

Class Name: TestSubjectDAOJpa	
Responsibilities: ▪ This class tests the SubjectDAO class with jUnit.	**Collaborations:** ▪ SubjectDAO: This is the class that is tested. ▪ Subject: used to manage fake-temporary object that we make in order to implement the tests.

Class Name: TestThesisDAOJpa	
Responsibilities: ▪ This class tests the ThesisDAO class with jUnit.	**Collaborations:** ▪ ThesisDAO: This is the class that is tested. ▪ Thesis: used to manage fake-temporary object that we make in order to implement the tests.

Class Name: TestUserDAOJpa	
Responsibilities: ▪ This class tests the UserDAO class with jUnit.	**Collaborations:** ▪ UserDAO: This is the class that is tested. ▪ User: used to manage fake-temporary object that we make in order to implement the tests.

Class Name: TestStrategies	
Responsibilities: ▪ This class tests the 4 algorithms that implement the strategy (in their own way) of the selection of the best applicant.	**Collaborations:** ▪ Student: To represent the applicants and the selected candidate. The following classes are used to call the methods that implement the strategy: ▪ BestGpaStrategy ▪ FewestRemainingCoursesStrategy ▪ RandomChoiseStrategy ▪ ThresholdStrategy

Class Name: TestProfessorService	
Responsibilities: ▪ This class tests all the methods of the ProfessorService interface with jUnit.	**Collaborations:** ▪ ProfessorService: This is the class that is tested. All the following classes are used to manage necessary fake-temporary objects that help with the testing. ▪ Professor, ProfessorDAO ▪ Subject, SubjectDAO ▪ Student, StudentDAO ▪ Application, ApplicationDAO ▪ Thesis, ThesisDAO ▪ ThesisAssignmentRequest

Class Name: TestStudentService	
Responsibilities: ▪ This class tests all the methods of the StudentService interface with jUnit.	**Collaborations:** ▪ StudentService: This is the class that is tested. All the following classes are used to manage necessary fake-temporary objects that help with the testing. ▪ Student, StudentDAO ▪ Application, ApplicationDAO ▪ Thesis, ThesisDAO

Class Name: TestSubjectService	
Responsibilities: ▪ This class tests all the methods of the SubjectService interface with jUnit.	**Collaborations:** ▪ SubjectService: This is the class that is tested. All the following classes are used to manage necessary fake-temporary objects that help with the testing. ▪ Subject, SubjectDAO ▪ Professor ▪ Student

Class Name: TestThesisService	
Responsibilities: ▪ This class tests all the methods of the ThesisService interface with jUnit.	**Collaborations:** ▪ ThesisService: This is the class that is tested. All the following classes are used to manage necessary fake-temporary objects that help with the testing. ▪ Thesis, ThesisDAO

Class Name: TestUserService	
Responsibilities: ▪ This class tests all the methods of the UserService interface with jUnit.	**Collaborations:** ▪ UserService: This is the class that is tested. All the following classes are used to manage necessary fake-temporary objects that help with the testing. ▪ User, UserDAO ▪ Role