Diploma Assignment

Sprint Report

ΙΩΑΝΝΗΣ ΦΟΥΡΝΑΡΗΣ ΑΜ:4828

ΒΑΣΙΛΕΙΟΣ ΙΩΑΝΝΗΣ ΜΠΟΥΖΑΜΠΑΛΙΔΗΣ ΑΜ:4744

VERSIONS HISTORY

Date	Version	Description	Author
19/5/2023	V1	Final project	Ioannis Fournaris, Vasileios Ioannis Bouzampalidis.
15/3/2023	V0	Implementation of use cases	Ioannis Fournaris, Vasileios Ioannis Bouzampalidis.

1 Introduction

This document provides information concerning the **<X>** sprint of the project.

1.1 Purpose

This project entails the development of a webpage utilizing HTML, Java, and Spring Boot tools. Its primary objective is to facilitate the assignment of diploma theses by professors based on students' progress at the university. The application offers a range of functionalities, including subject management, their display, handling student applications, their display, thesis management and display, and profile information editing. These features collectively aim to streamline the process of thesis assignment and enhance the overall efficiency of the system.

1.2 Document Structure

The rest of this document is structured as follows. Section 2 describes out Scrum team and specifies this 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	....ΙΩΑΝΝΗΣ ΦΟΥΡΝΑΡΗΣ, ΒΑΣΙΛΕΙΟΣ ΙΩΑΝΝΗΣ ΜΠΟΥΖΑΜΠΑΛΙΔΗΣ
Scrum Master	.... ΒΑΣΙΛΕΙΟΣ ΙΩΑΝΝΗΣ ΜΠΟΥΖΑΜΠΑΛΙΔΗΣ
Development Team	....ΙΩΑΝΝΗΣ ΦΟΥΡΝΑΡΗΣ, ΒΑΣΙΛΕΙΟΣ ΙΩΑΝΝΗΣ ΜΠΟΥΖΑΜΠΑΛΙΔΗΣ

2.2 Sprints

<List below the sprints that you performed and the user stories that have been realized in each Sprint>

Sprint No	Begin Date	End Date	Number of weeks	User stories
1	**30/3**	**7/4**	**1**	**All "user" stories**
2	**15/4**	**30/4**	**2**	**All "student" stories**
3	**5/5**	**19/5**	**2**	**All "professor" stories**

3 Use Cases

<Specify the concrete Use Cases that describe the interaction of the user with the applications, as derived from the abstract user stories. Give a **UML Use Case diagram** and the **detailed use case descriptions**.>

<Use Case 1>(User)

3.1 UC1: REGISTER

Use case ID	U_1
Actors	User
Pre conditions	The user is not already registered in the application.
Main flow of events	[Main flow of events that describes the interaction between the user and the application] 1. The use case starts when the user selects the **"Register"** option on the application's Register page. 2. The application presents a registration form for the user to fill out his role(student or professor), username and password.

	3. The application verifies that the username is not already taken. 4. The application creates a new account for the user with the specified role, username, and password. 5. The application redirects the user to the login page.
Alternative flow 1	If the username is already taken, the application displays an error message, prompts the user to enter a different username and redirects to the Register page.
Alternative flow 2	If the user fails to enter the required fields or enters invalid data, the application redirects to the login page for the user to correct their input.
Post conditions	The user's account is created and the user can now login to the application using the appropriate credentials.

<Use Case 2>(User)

3.2 UC2: LOGIN

Use case ID	U_2
Actors	User
Pre conditions	1. 1.The user must have a non–expired username and password 2. The user's account must exist in the application
Main flow of events	[Main flow of events that describes the interaction between the user and the application] 1. The use case starts when the user navigates to the login page. 2. The user enters their username and password. 3. The Application validates the credentials and logs in the user by selecting the **"Login"** option. 4. The Application presents the Homepage depending on the user's role.
Alternative flow 1	If the user fails to enter the required fields or enters invalid data, the application redirects to the login page for the user to correct their input.
Alternative flow 2	-
Post conditions	The user has access to the provided functionalities.

3.1 UC1: EDIT CRECENTIALS

Use case ID	S_1
Actors	Student
Pre conditions	1. The student is logged into the application
Main flow of events	[Main flow of events that describes the interaction between the user and the application] 1. The use case starts when the student navigates to the Edit Credentials page by selecting the **"Edit Credentials"** option. 2. The student enters their full name, year of studies, current average grade, and number of remaining courses for graduation in the appropriate fields. 3. The application saves the student's profile information by selecting the **"Save Profile"** option. 4. The application redirects the student to the Student Homepage.
Alternative flow 1	-
Alternative flow 2	-
Post conditions	The professors can access the student's profile information for evaluation of their application to available diploma thesis subjects.

3.2 UC2: VIEW ALL SUBJECTS

Use case ID	S_2
Actors	Student
Pre conditions	The student is logged into the application
Main flow of events	[Main flow of events that describes the interaction between the user and the application] 1. The use case starts when the student navigates to the View All Subjects page. 2. The application displays a list of available diploma thesis subjects offered by the professors by selecting the **"View All Subjects"** option. 3. The application provides the option to the student to select one or more diploma thesis subjects that look interesting by selecting the **"Apply To Subject"** option. 4. The application provides a detailed description of each available diploma thesis subject in the column "objectives".
Alternative flow 1	-
Alternative flow 2	-
Post conditions	The student has viewed the available diploma thesis subjects.

<Use Case 3>(Student)

3.3 UC3: APPLY TO SUBJECT

Use case ID	S_3
Actors	Student
Pre conditions	The student must have viewed a more detailed description of the thesis subject they wish to apply for.
Main flow of events	[Main flow of events that describes the interaction between the user and the application] 1. The use case starts when the student navigates to the detailed description of the subject that chose to apply for. 2. The student selects a specific subject of interest. 3. The student selects the **"Apply To Subject"** option.
Alternative flow 1	-
Alternative flow 2	-
Post conditions	1. The student has applied for a diploma thesis. 2. The student has notified the professor for the interest in taking over the thesis.

<Use Case 4>(Student)

3.4 UC4: LOGOUT

Use case ID	S_4
Actors	Student
Pre conditions	The student is logged in the application.
Main flow of events	[Main flow of events that describes the interaction between the user and the application] 1. The use case starts when the Student selects the **"Logout"** option.

	2. The application initiates a logout process for the student and disconnects them from their account. 3. The application redirects the student to the login page.
Alternative flow 1	-
Alternative flow 2	-
Post conditions	The application has disconnected the student.

<Use Case 1 > (Professor)

3.1 UC1: EDIT CREDENTIALS

Use case ID	P_1
Actors	Professor
Pre conditions	The professor has logged in the application.
Main flow of events	[Main flow of events that describes the interaction between the user and the application] 1. The use case starts when the professor navigates to the Edit Credentials page by selecting the **"Edit Credentials"** option. 2. The professor enters their full name, specialty and threshold values in the appropriate fields. 3. The application saves the professor's profile information by selecting the **"Save Profile"** option. 4. The application redirects the professor to the Professor Homepage.
Alternative flow 1	-
Alternative	-

flow 2	
Post conditions	The students can access the professor's profile information in order to get informed about them, before applying to available diploma thesis subjects.

<Use Case 2 > (Professor)

3.2 UC2: MY SUBJECTS

Use case ID	P_2
Actors	Professor
Pre conditions	The professor has logged in the application.
Main flow of events	[Main flow of events that describes the interaction between the user and the application] 1. The use case starts when the professor navigates to the My Subjects section of the system by selecting the **"My Subjects"** option. 2. The application displays a list of available diploma subjects offered by the professor. 3. The professor selects the diploma subjects to manage. 4. The application displays detailed information about the selected diploma thesis subject, including the title, objectives, and the names of students who have applied for the subject.
Alternative flow 1	-
Alternative flow 2	-
Post conditions	The professor has accessed the available diploma thesis subjects for management.

3.3 UC3: ADD SUBJECT

Use case ID	P_3
Actors	Professor
Pre conditions	The professor has access to the available list of diploma subjects.
Main flow of events	[Main flow of events that describes the interaction between the user and the application] 1. The use case starts when the professor selects the **"Add Subject"** option. 2. The professor enters the new subject's attributes (title, objectives). 3. The application adds the new subject to the list of available diploma subjects by selecting the **"Save Subject"** option.
Alternative flow 1	-
Alternative flow 2	-
Post conditions	The application has updated the list of available diploma subjects with the new subject.

3.4 UC4: DELETE SUBJECT

Use case ID	P_4
Actors	Professor
Pre conditions	The professor has access to the available list of diploma subjects.

Main flow of events	[Main flow of events that describes the interaction between the user and the application] 1. The use case starts when the professor selects the **"Delete Subject"** option. 2. The application displays a page for the professor to enter the subject's title they want to delete. 3. The application deletes the selected subject from the list of available diploma subjects by selecting the **"Delete Subject"** option.
Alternative flow 1	If the professor cancels the delete operation, the application does not delete the selected diploma thesis subject.
Alternative flow 2	-
Post conditions	The application has updated the list of available diploma subjects without the selected subject.

<Use Case 5 > (Professor)

3.5 UC5: APPLICATIONS

Use case ID	P_5
Actors	Professor
Pre conditions	1. The professor has access to the available list of diploma subjects. 2. At least one student has applied for the selected diploma thesis.
Main flow of events	[Main flow of events that describes the interaction between the user and the application] 1. The use case starts when the professor navigates to the Applications by selecting the **"Applications"** option. 2. The professor selects the diploma subject they wish to view applications for by entering the title of the subject. 3. The application displays a list of all student applications for the

	selected diploma subject. 4. The professor reviews the applications for the diploma thesis.
Alternative flow 1	If no student has applied for the selected diploma subject, the application displays an empty list.
Alternative flow 2	-
Post conditions	The professor has viewed student's applications.

<Use Case 6 > (Professor)

3.6 UC6: RANDOM CHOICE STRATEGY

Use case ID	P_6
Actors	Professor
Pre conditions	The professor has access to the list of students who have applied for the diploma thesis project.
Main flow of events	[Main flow of events that describes the interaction between the user and the application] 1. The use case starts when the professor enters the **"Random"** option corresponding to the random choice strategy to assign the diploma subject. 2. The application randomly selects a student from the list of applicants. 3. The application assigns the diploma thesis to the selected student.
Alternative flow 1	If there are no applicants for the diploma thesis subject, the application displays an error message indicating that there are no applicants.
Alternative flow 2	-
Post conditions	The application has assigned the diploma thesis to the student randomly.

3.7 UC7: BEST AVERAGE GRADE STRATEGY

Use case ID	P_7
Actors	Professor
Pre conditions	The professor has access to the list of students who have applied for the diploma thesis project.
Main flow of events	[Main flow of events that describes the interaction between the user and the application] 1. The use case starts when the professor enters the **"Best Average Grade"** option corresponding to the best average grade strategy to assign the diploma subject. 2. The application compares the average grade of each applicant. 3. The application selects the student with the highest average grade. 4. The application assigns the diploma thesis to the selected student.
Alternative flow 1	If there are no applicants for the diploma thesis subject, the application displays an error message indicating that there are no applicants.
Alternative flow 2	If there are multiple applicants with the same average course grade, the application assigns the thesis randomly.
Post conditions	The application has assigned the diploma thesis subject to the student based on the best average grade.

3.8 UC8: FEWEST COURSES STRATEGY

Use case ID	P_8
Actors	Professor
Pre conditions	The professor has access to the list of students who have applied for the diploma thesis project.
Main flow of events	[Main flow of events that describes the interaction between the user and the application] 1. The use case starts when the professor enters **"Fewest Courses"** option corresponding to the fewest courses strategy to assign the diploma subject. 2. The application compares the number of remaining courses for each applicant. 3. The application selects the student with the fewest remaining courses. 4. The application assigns the diploma thesis to the selected student.
Alternative flow 1	If there are no applicants for the diploma thesis subject, the application displays an error message indicating that there are no applicants.
Alternative flow 2	If there are multiple applicants with the same number of courses for graduation, the application assigns the thesis randomly.
Post conditions	The application has assigned the diploma thesis subject to the student based on the remaining courses for graduation.

3.9 UC9: THRESHOLD STRATEGY

Use case ID	P_9
Actors	Professor
Pre conditions	The professor has access to the list of students who have applied for the diploma thesis project.
Main flow of events	[Main flow of events that describes the interaction between the user and the application] 1. The use case starts when the professor enters **"Threshold"** option corresponding to the threshold strategy to assign the diploma subject. 2. The application filters the list of students based on the entered threshold values 3. The application selects the student with the appropriate threshold values. 4. The application assigns the diploma thesis to the selected student.
Alternative flow 1	If there are no applicants for the diploma thesis subject, the application displays an error message indicating that there are no applicants.
Alternative flow 2	If there are multiple applicants with the same threshold values, the application assigns the thesis randomly.
Alternative flow 3	If the professor didn't enter the threshold values in his credentials, the application displays an error message and prompts the professor to fill the threshold values.
Post conditions	The application has assigned the diploma thesis subject to the student based on threshold values.

<Use Case 10 > (Professor)

3.10 UC10: MY THESIS

Use case ID	P_10
Actors	Professor
Pre conditions	The professor is assigned as a supervisor for at least one diploma thesis project.

Main flow of events	[Main flow of events that describes the interaction between the user and the application] 1. The use case starts when the Professor selects the **"My Thesis"** option. 2. The application displays a list of all diploma thesis assigned to the student. 3. The professor can edit or add information related to the project.
Alternative flow 1	-
Alternative flow 2	-
Post conditions	The Professor has accessed the list of Diploma Thesis.

Use Case 11 > (Professor)

3.11 UC11: SET GRADES FOR PROJECT

Use case ID	P_11
Actors	Professor
Pre conditions	The professor must have access to their thesis.
Main flow of events	[Main flow of events that describes the interaction between the user and the application] 1. The use case starts when the Professor selects the **"Set Grade"** option. 2. The Professor selects the project for which they want to set grades. i. The Professor sets grades for the implementation by entering the **"Impl Grade"** value. ii. The Professor sets grades for the report by entering the **"Report Grade"** value. iii. The Professor sets grades for the presentation by

	entering the **"Presentation Grade"** value. 3. The application saves the grades.
Alternative flow 1	-
Alternative flow 2	-
Post conditions	The application has saved the grades.

3.12 UC12: SAVE GRADE

Use case ID	P_12
Actors	Professor
Pre conditions	The implementation, report, and presentation grades must have been set for the diploma thesis.
Main flow of events	[Main flow of events that describes the interaction between the user and the application] 4. The use case starts when the Professor selects the **"Save Grade"** option. 5. The application calculates the overall grade using the formula: total grade = 0.7 * implementation grade + 0.15 * report grade + 0.15 * presentation grade. 6. The application saves the overall grade. 7. The application updates the My Thesis list.

Alternative flow 1	-
Alternative flow 2	-
Post conditions	The application has saved the overall grade.

Use Case 13 > (Professor)

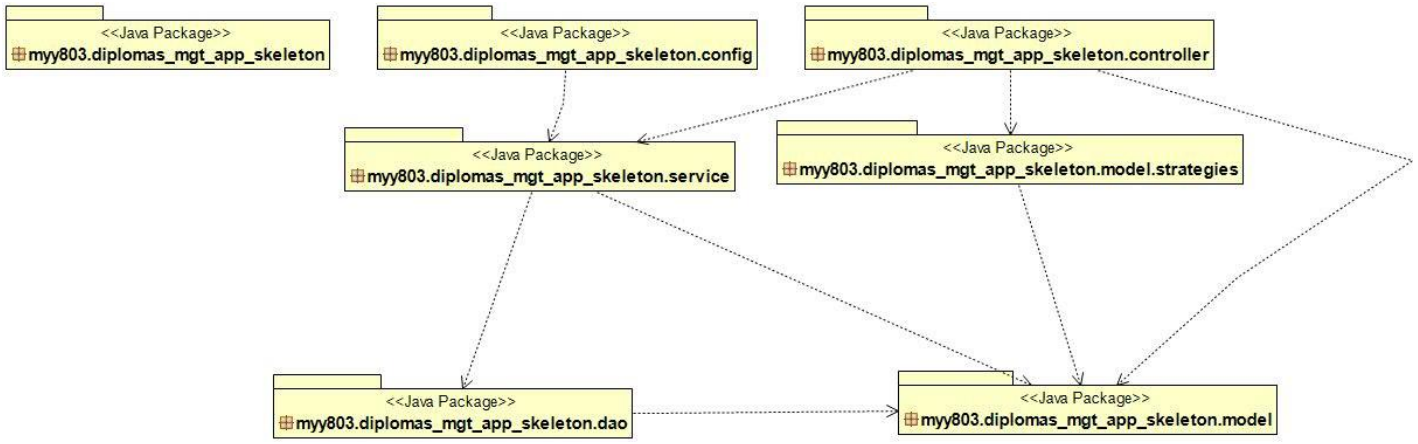
3.13 UC13: LOGOUT

Use case ID	P_12
Actors	Professor
Pre conditions	The professor is logged in the application.
Main flow of events	[Main flow of events that describes the interaction between the user and the application] 8. The use case starts when the Professor selects the **"Logout"** option. 9. The application initiates a logout process for the professor and disconnects them from their account. 10. The application redirects the professor to the login page.
Alternative flow 1	-
Alternative flow 2	-
Post conditions	The application has disconnected the professor.

4 Design

4.1 Architecture

<Specify the overall architecture for this release in terms of a **UML package diagram**.>



4.2 Design

<Specify the detailed design for this release in terms of **UML class diagrams**.>

<Document the classes that are included in this release in terms of CRC cards according to the template that is given below.>

DAO PACKAGE

Interface Name: ApplicationDAO	
Responsibilities: ▪ An interface that provides access to the database ▪ Define data access operations related to the Application entity. ▪ Inherit the basic CRUD operations from the JpaRepository interface.	**Collaborations:** ▪ ApplicationDAO collaborates with the ApplicationServiceImpl to implement the service logic for the Application entity. ▪ It also collaborates with the JpaRepository interface, which provides the basic database operations implementation for the Application entity.

Interface Name: ProfessorDAO	
Responsibilities: ▪ An interface that provides access to the database ▪ Define data access operations related to the Professor entity. ▪ Inherit the basic CRUD operations from the JpaRepository interface.	**Collaborations:** ▪ ProfessorDAO collaborates with the ProfessorServiceImpl to implement the service logic for the Professor entity. ▪ It also collaborates with the JpaRepository interface, which provides the basic database operations implementation for the Professor entity.

Interface Name: StudentDAO	
Responsibilities: ▪ An interface that provides access to the database ▪ Define data access operations related to the Student entity. ▪ Inherit the basic CRUD operations from the JpaRepository interface.	**Collaborations:** ▪ StudentDAO collaborates with the StudentServiceImpl to implement the service logic for the Student entity. ▪ It also collaborates with the JpaRepository interface, which provides the basic database operations implementation for the Student entity.

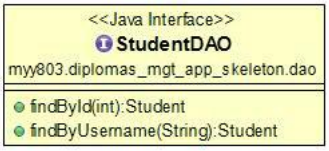
Interface Name: SubjectDAO	
Responsibilities: ▪ An interface that provides access to the database ▪ Define data access operations related	**Collaborations:** ▪ SubjectDAO collaborates with the SubjectServiceImpl to implement the service logic for the Subject entity.

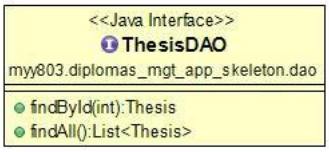
to the Subject entity. ▪ Inherit the basic CRUD operations from the JpaRepository interface.	▪ It also collaborates with the JpaRepository interface, which provides the basic database operations implementation for the Subject entity.

Interface Name: ThesisDAO	
Responsibilities: ▪ An interface that provides access to the database ▪ Define data access operations related to the Thesis entity. ▪ Inherit the basic CRUD operations from the JpaRepository interface.	**Collaborations:** ▪ ThesisDAO collaborates with the ThesisServiceImpl to implement the service logic for the Thesis entity. ▪ It also collaborates with the JpaRepository interface, which provides the basic database operations implementation for the Thesis entity.

Interface Name: UserDAO	
Responsibilities: ▪ An interface that provides access to the database ▪ Define data access operations related to the User entity. ▪ Inherit the basic CRUD operations from the JpaRepository interface.	**Collaborations:** ▪ UserDAO collaborates with the UserServiceImpl to implement the service logic for the User entity. ▪ It also collaborates with the JpaRepository interface, which provides the basic database operations implementation for the User entity.

UML:





MODEL PACKAGE

Class Name: Application	
Responsibilities: ▪ Represents an application for a subject made by a student and assigned to a professor. ▪ Contains fields for storing references to related entities such as Student, Subject, and Professor	**Collaborations:** ▪ Application class collaborates with the Student class, which represents the student making the application ▪ It collaborates with the Subject class, which represents the subject for which the application is made. ▪ It also collaborates with the Professor class, which represents the professor assigned to the application.

Class Name: Professor

Responsibilities:	Collaborations:
▪ Represents a professor in the system. ▪ Contains fields for storing information about the professor such as fullName, specialty, professorId, username, threshold1, and threshold2. ▪ Provides getter and setter methods for accessing and modifying the class properties.	▪ It has association with the User entity because a foreign key 'username' is declared that corresponds to the username field from User.

Class Name: Student	
Responsibilities:	**Collaborations:**
▪ Represents a student in the system. ▪ Contains fields for storing information such as fullName, yearOfStudies, currentAverageGrade, numberOfRemainingCoursesForGraduation, studentId, and username ▪ Provides getter and setter methods for accessing and modifying the class properties.	▪ It has association with the User entity because a foreign key 'username' is declared that corresponds to the username field from User.

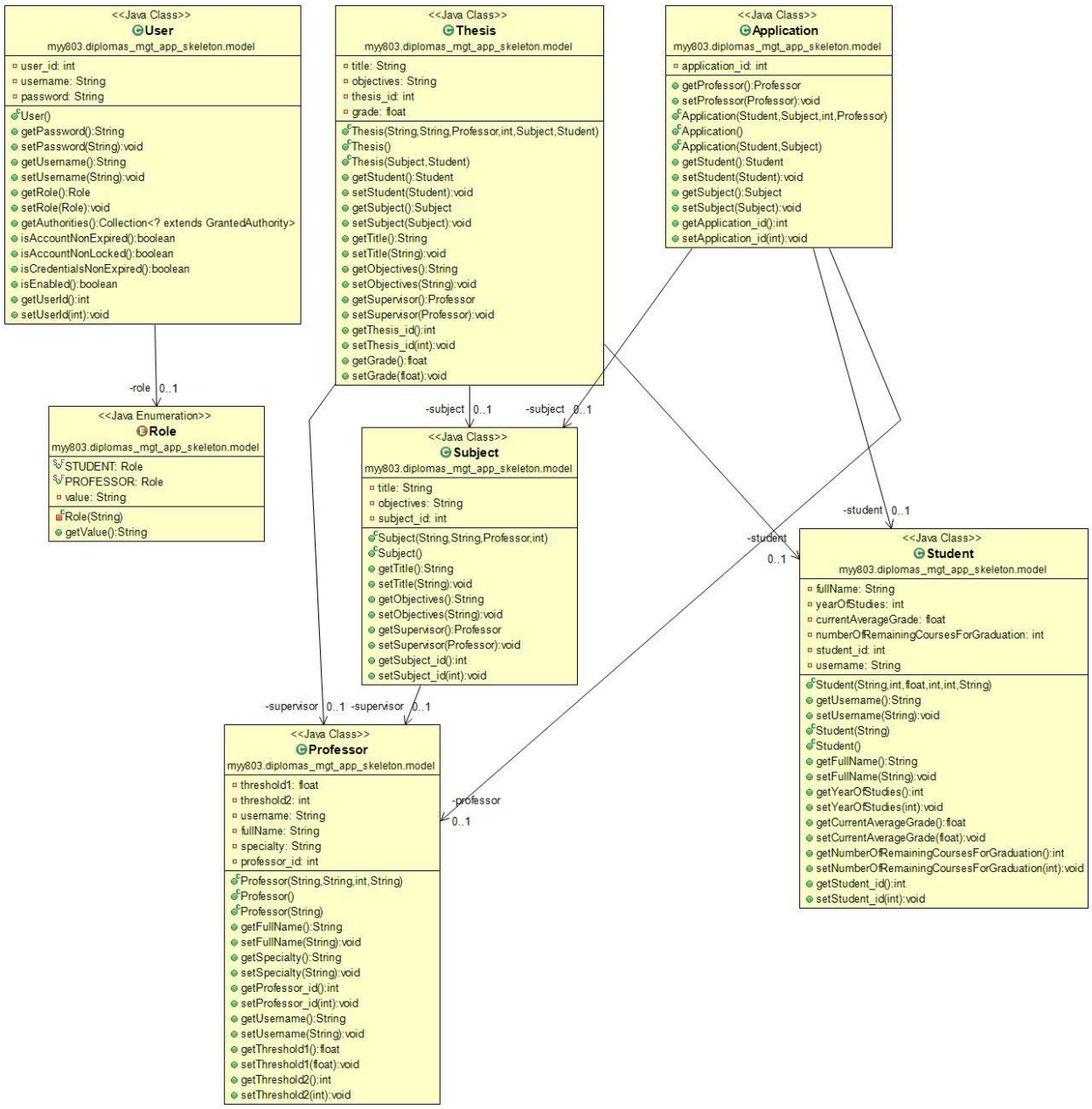
Class Name: Subject	
Responsibilities:	**Collaborations:**
▪ Represents a subject in the system. ▪ Contains fields for storing information such as title, objectives, supervisor, and subjectId. ▪ Provides getter and setter methods for accessing and modifying the class properties.	▪ The Subject class collaborates with the Professor class, which represents the supervisor assigned to the subject.

Class Name: Thesis

Responsibilities:	**Collaborations:**
▪ Represents a thesis in the system. ▪ Contains fields for storing information such as title, objectives, supervisor, subject, student, studentId, subjectId, , grade and thesisId. ▪ Provides getter and setter methods for accessing and modifying the class properties.	▪ The Thesis class collaborates with the Professor class, which represents the supervisor of the thesis. ▪ It also collaborates with the Subject class, which represents the subject to which the thesis is related. ▪ The Thesis class collaborates with the Student class, which represents the student working on the thesis

Class Name: User	
Responsibilities:	**Collaborations:**
▪ Represents a User in the system. ▪ Implements the UserDetails interface from the Spring Security framework, which provides user-related security information. ▪ Contains fields for storing information such as password, username, and role. ▪ Provides getter and setter methods for accessing and modifying the class properties.	▪ The User class collaborates with the Spring Security framework by implementing the UserDetails interface. ▪ It uses the Role enum to define the user's role

UML:


<<Java Class>>
User
myy803.diplomas_mgt_app_skeleton.model
user_id: int
username: String
password: String
User()
getPassword():String
setPassword(String):void
getUsername():String
setUsername(String):void
getRole():Role
setRole(Role):void
getAuthorities():Collection<? extends GrantedAuthority>
isAccountNonExpired():boolean
isAccountNonLocked():boolean
isCredentialsNonExpired():boolean
isEnabled():boolean
getUserId():int
setUserId(int):void
<<Java Class>>
Thesis
myy803.diplomas_mgt_app_skeleton.model
title: String
objectives: String
thesis_id: int
grade: float
Thesis(String,String,Professor,int,Subject,Student)
Thesis()
Thesis(Subject,Student)
getStudent():Student
setStudent(Student):void
getSubject():Subject
setSubject(Subject):void
getTitle():String
setTitle(String):void
getObjectives():String
setObjectives(String):void
getSupervisor():Professor
setSupervisor(Professor):void
getThesis_id():int
setThesis_id(int):void
getGrade():float
setGrade(float):void
<<Java Class>>
Application
myy803.diplomas_mgt_app_skeleton.model
application_id: int
getProfessor():Professor
setProfessor(Professor):void
Application(Student,Subject,int,Professor)
Application()
Application(Student,Subject)
getStudent():Student
setStudent(Student):void
getSubject():Subject
setSubject(Subject):void
getApplication_id():int
setApplication_id(int):void
-role 0..1
<<Java Enumeration>>
Role
myy803.diplomas_mgt_app_skeleton.model
STUDENT: Role
PROFESSOR: Role
value: String
Role(String)
getValue():String
-subject 0..1
-subject 0..1
<<Java Class>>
Subject
myy803.diplomas_mgt_app_skeleton.model
title: String
objectives: String
subject_id: int
Subject(String,String,Professor,int)
Subject()
getTitle():String
setTitle(String):void
getObjectives():String
setObjectives(String):void
getSupervisor():Professor
setSupervisor(Professor):void
getSubject_id():int
setSubject_id(int):void
-student 0..1
-student 0..1
<<Java Class>>
Student
myy803.diplomas_mgt_app_skeleton.model
fullName: String
yearOfStudies: int
currentAverageGrade: float
numberOfRemainingCoursesForGraduation: int
student_id: int
username: String
Student(String,int,float,int,int,String)
getUsername():String
setUsername(String):void
Student(String)
Student()
getFullName():String
setFullName(String):void
getYearOfStudies():int
setYearOfStudies(int):void
getCurrentAverageGrade():float
setCurrentAverageGrade(float):void
getNumberOfRemainingCoursesForGraduation():int
setNumberOfRemainingCoursesForGraduation(int):void
getStudent_id():int
setStudent_id(int):void
-supervisor 0..1
-supervisor 0..1
<<Java Class>>
Professor
myy803.diplomas_mgt_app_skeleton.model
threshold1: float
threshold2: int
username: String
fullName: String
specialty: String
professor_id: int
Professor(String,String,int,String)
Professor()
Professor(String)
getFullName():String
setFullName(String):void
getSpecialty():String
setSpecialty(String):void
getProfessor_id():int
setProfessor_id(int):void
getUsername():String
setUsername(String):void
getThreshold1():float
setThreshold1(float):void
getThreshold2():int
setThreshold2(int):void
-professor
0..1

SERVICE PACKAGE

Class Name: ApplicationServiceImpl	
Responsibilities: ▪ Implements the ApplicationService interface, which defines the service methods related to the Application entity. ▪ Autowires the ApplicationDAO dependency to interact with the database ▪ Provides implementations for the methods defined in the ApplicationService interface: 1) save(Application application): Saves the given Application object by invoking the corresponding method in the ApplicationDAO. 2) findAll(): Retrieves a list of all Application objects from the database by invoking the corresponding method in the ApplicationDAO. 3) findById(int applicationId): Retrieves an Application object by its unique applicationId from the database by invoking the corresponding method in the ApplicationDAO.	**Collaborations:** ▪ Collaborates with the ApplicationDAO interface to perform database operations related to the Application entity. ▪ Collaborates with the Application model class for working with Application objects. ▪ Depends on the Spring framework as it uses the Autowired annotation

Class Name: ProfessorServiceImpl	
Responsibilities: ▪ Implements the ProfessorService interface, which defines the service methods related to the Professor entity. ▪ Autowires the ProfessorDAO, SubjectDAO, ThesisDAO, and ApplicationDAO dependencies to interact with the database for CRUD operations on corresponding objects. ▪ Provides implementations for the methods defined in the ProfessorService interface ▪ Maintains lists (professorSubjects, professorThesis, professorApplications) to store the objects for efficient access.	**Collaborations:** ▪ Collaborates with the ProfessorDAO, SubjectDAO, ThesisDAO, and ApplicationDAO interfaces to perform database operations related to the corresponding entities. ▪ Collaborates with the Professor, Subject, Thesis, Application, and Student model classes for working with objects. ▪ Depends on the Spring framework as it uses the Autowired annotation

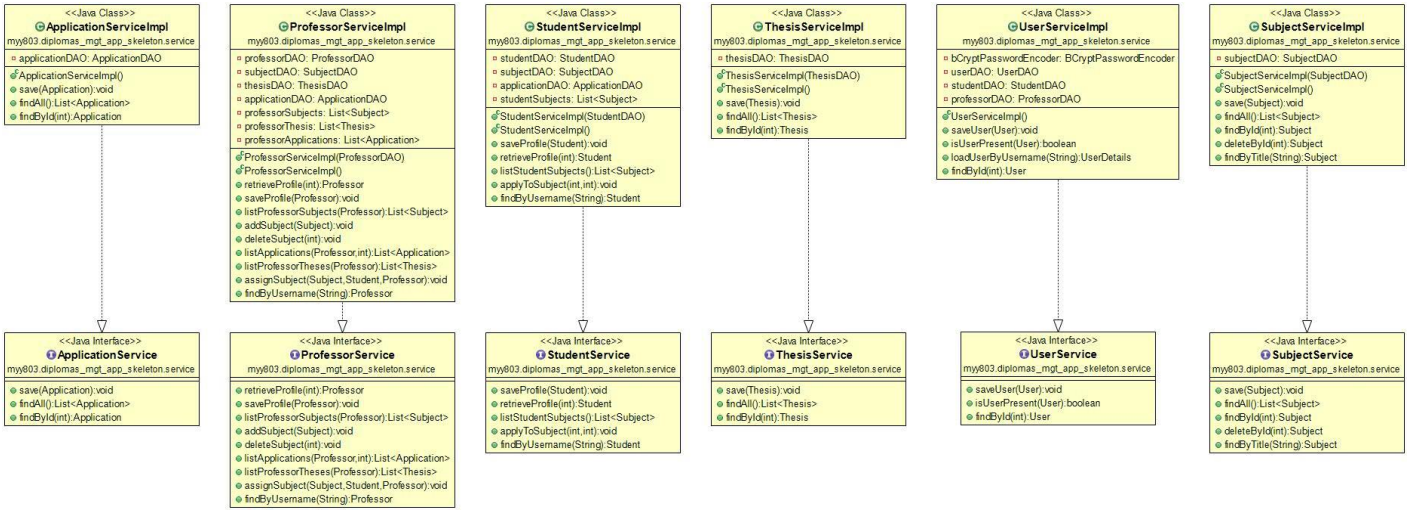
Class Name: StudentServiceImpl	
Responsibilities: ▪ Implements the StudentService interface, which defines the service methods related to the Student entity. ▪ Autowires the StudentDAO, SubjectDAO, and ApplicationDAO dependencies to interact with the database ▪ Provides implementations for the methods defined in the StudentService interface ▪ Maintains a list (studentSubjects) to store the retrieved Subject objects for efficient access.	**Collaborations:** ▪ Collaborates with the StudentDAO, SubjectDAO, and ApplicationDAO interfaces ▪ Collaborates with the Student, Subject, Application, and Professor model classes for working with objects. ▪ Depends on the Spring framework as it uses the Autowired annotation

Class Name: SubjectServiceImpl	
Responsibilities: ▪ Implements the SubjectService interface, which defines the service methods related to the Subject entity. ▪ Autowires the SubjectDAO dependency to interact with the database. ▪ Provides implementations for the methods defined in the SubjectService interface	**Collaborations:** ▪ Collaborates with the SubjectDAO interface to perform database operations related to the Subject entity. ▪ Collaborates with the Subject model class for working with Subject objects. ▪ Depends on the Spring framework as it uses the Autowired annotation

Class Name: ThesisServiceImpl	
Responsibilities: ▪ Implements the ThesisService interface, which defines the service methods related to the Thesis entity. ▪ Autowires the ThesisDAO dependency to interact with the database for CRUD operations on Thesis objects. ▪ Provides implementations for the methods defined in the ThesisService interface	**Collaborations:** ▪ Collaborates with the ThesisDAO interface and its implementation to perform database operations related to the Thesis entity. ▪ Collaborates with the Thesis model class for working with Thesis objects. ▪ Depends on the Spring framework for autowiring the ThesisDAO bean.

Class Name: UserServiceImpl	
Responsibilities: ▪ Implements the UserService interface, which defines the service methods related to user management. ▪ Implements the Spring UserDetailsService interface to load user details. ▪ Autowires the necessary dependencies, including BCryptPasswordEncoder, UserDAO, StudentDAO, and ProfessorDAO. ▪ Provides implementations for the methods defined in the UserService interface	**Collaborations:** ▪ Collaborates with the UserDAO interface to perform database operations related to user entities. ▪ Collaborates with the StudentDAO and ProfessorDAO interfaces and their implementations to create and save Student and Professor entities based on the user's role. ▪ Depends on the Spring framework for autowiring the necessary beans, such as BCryptPasswordEncoder, UserDAO, StudentDAO, and ProfessorDAO. ▪ Collaborates with the User model class for working with user objects.

UML:



STRATEGIES PACKAGE

Class Name: BestApplicantStrategyFactory	
Responsibilities: ▪ Provides a factory method createStrategy(String selection) for creating instances of different strategies based on the given selection parameter. ▪ Depending on the value of the selection parameter, the factory method creates and returns an instance of the corresponding strategy. ▪ The factory method initializes the selectedStrategy variable with the corresponding strategy instance based on the selection.	**Collaborations:** ▪ Collaborates with the BestApplicantStrategy interface and its concrete strategy implementations, such as BestAvgGradeStrategy, FewestCoursesStrategy, RandomChoiceStrategy, and ThresholdStrategy.

Class Name: BestAvgGradeStrategy	
Responsibilities: ▪ Represents the strategy for selecting the best applicant based on the highest average grade. ▪ Overrides the compareApplications method defined in the TemplateStrategyAlgorithm class. ▪ Compares two Application objects based on the average grades of their corresponding students. ▪ Determines the student ID of the applicant with the higher average grade.	**Collaborations:** ▪ Collaborates with the Application class and its Student objects to retrieve the average grades of the applicants. ▪ Inherits from the TemplateStrategyAlgorithm class, which defines the template and common behavior for strategy algorithms.

▪ Returns the student ID of the selected applicant. ▪ If the average grades are equal, it randomizes the applicants	

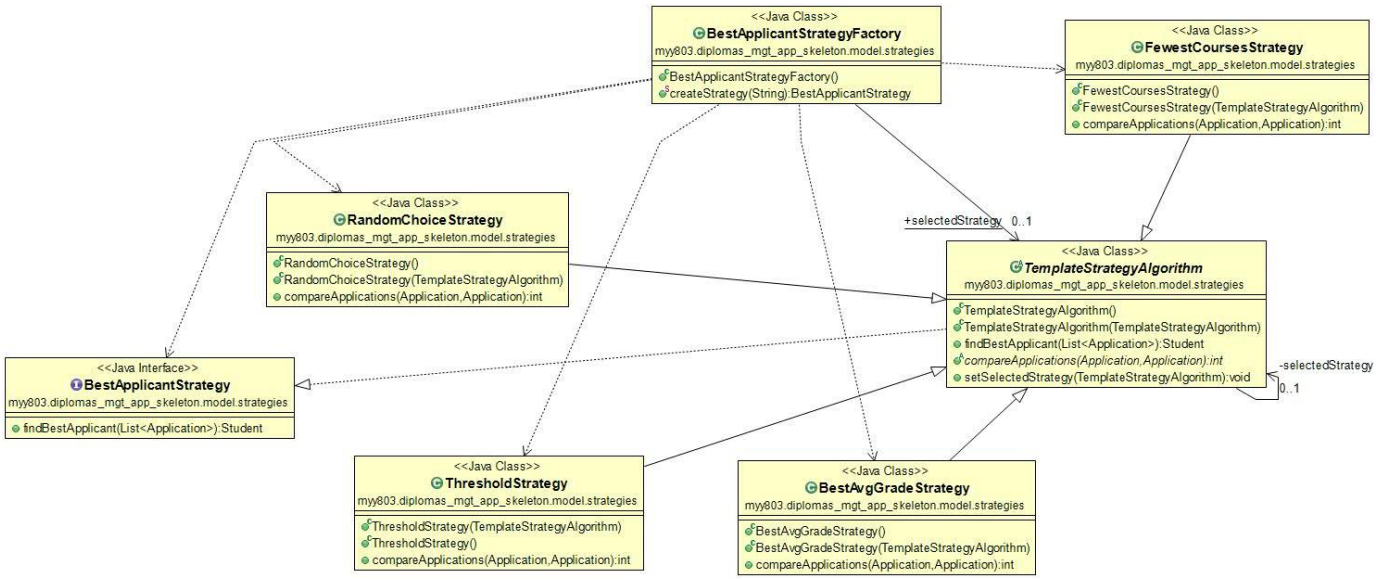
Class Name: FewestCoursesStrategy	
Responsibilities: ▪ Represents the strategy for selecting the best applicant based on the fewest remaining courses for graduation. ▪ Overrides the compareApplications method defined in the TemplateStrategyAlgorithm class. ▪ Compares two Application objects based on the number of remaining courses for graduation of their corresponding students. ▪ Determines the student ID of the applicant with the fewest remaining courses. ▪ Returns the student ID of the selected applicant. ▪ If the number of remaining courses is equal, it randomizes the applicants	**Collaborations:** ▪ Collaborates with the Application class and its Student objects to retrieve the number of remaining courses for graduation of the applicants. ▪ Inherits from the TemplateStrategyAlgorithm class, which defines the template and common behavior for strategy algorithms.

Class Name: ThresholdStrategy	
Responsibilities: ▪ Represents the strategy for selecting the best applicant based on a threshold condition. ▪ Overrides the compareApplications method defined in the TemplateStrategyAlgorithm class. ▪ Compares the current average grade and the number of remaining courses of the applicants with the respective threshold values. ▪ If both applicants satisfy the threshold condition, it randomizes them. ▪ If the first applicant satisfies the threshold condition, the student ID of the first applicant is returned. ▪ If the second applicant satisfies the threshold condition, the student ID of the second applicant is returned. ▪ If neither applicant fulfills the threshold condition, the method returns the value of -1 to facilitate the continued looping in order to find an applicant that satisfies the threshold condition.	**Collaborations:** ▪ Collaborates with the Application class and its Student and Professor objects to retrieve the relevant information for comparison. ▪ Inherits from the TemplateStrategyAlgorithm class, which defines the template and common behavior for strategy algorithms.

Class Name: TemplateStrategyAlgorithm	
Responsibilities: ▪ Provides a template for strategy algorithms for selecting the best student. ▪ Contains a reference to the selected strategy algorithm (selectedStrategy) of type TemplateStrategyAlgorithm. ▪ Defines an abstract method compareApplications to be implemented by concrete strategy classes. ▪ Defines a method findBestApplicant that takes a list of Application objects and returns the best Student based on the selected strategy. ▪ If the selected strategy is an instance of RandomChoiceStrategy, a random student is selected from the list. ▪ If the selected strategy is not RandomChoiceStrategy, the comparison is performed between the first application and each subsequent application in the list using the compareApplications method. ▪ Returns the best student based on the comparison results.	**Collaborations:** ▪ Collaborates with the Application class and its Student object to retrieve the relevant information for comparison. ▪ Inherits from the BestApplicantStrategy interface, defining the method compareApplications. ▪ Works in conjunction with concrete strategy classes such as BestAvgGradeStrategy, FewestCoursesStrategy, and ThresholdStrategy that extend the TemplateStrategyAlgorithm class and provide the implementation for the compareApplications method.

Class Name: RandomChoiceStrategy	
Responsibilities: ▪ The class selects a random applicant from the list of applications and designates it as the bestApplicant.	**Collaborations:** ▪ Inherits from the TemplateStrategyAlgorithm class, which defines the template and common behavior for strategy algorithms.

UML:



CONTROLLER PACKAGE

Class Name: AuthController	
Responsibilities: ▪ Handle user authentication and authorization functionalities. ▪ Handle user registration and saving user information. ▪ Manage user interface interactions related to authentication and registration. ▪ Redirect users to the login page or handle login/logout actions. ▪ Render appropriate views and messages based on the authentication and registration process.	**Collaborations:** ▪ UserService: Collaborates with UserService to perform user-related operations such as checking if a user is already registered and saving user information. ▪ StudentService: Collaborates with StudentService to access student-related functionalities if needed. ▪ Model classes (Role, Student, User): Uses model classes to represent user roles, student information, and user information. ▪ Spring Framework: Utilizes various Spring annotations and features for handling requests, dependency injection, and rendering views ▪ Model interface (Model): Collaborates with the Model interface to pass data between the controller and the view. ▪ View templates (auth/login, auth/register, auth/error): Renders the appropriate view templates to display the login, registration, and error pages.

Class Name: ProfessorController	
Responsibilities: ▪ Handle professor-related functionalities and interactions. ▪ Manage professor's subjects, theses, applications, and profile. ▪ Render appropriate views and handle requests for professor-related	**Collaborations:** ▪ ProfessorService: Collaborates with ProfessorService to perform operations related to professor entities such as retrieving, saving, and listing professor information. ▪ SubjectService: Collaborates with SubjectService to perform operations

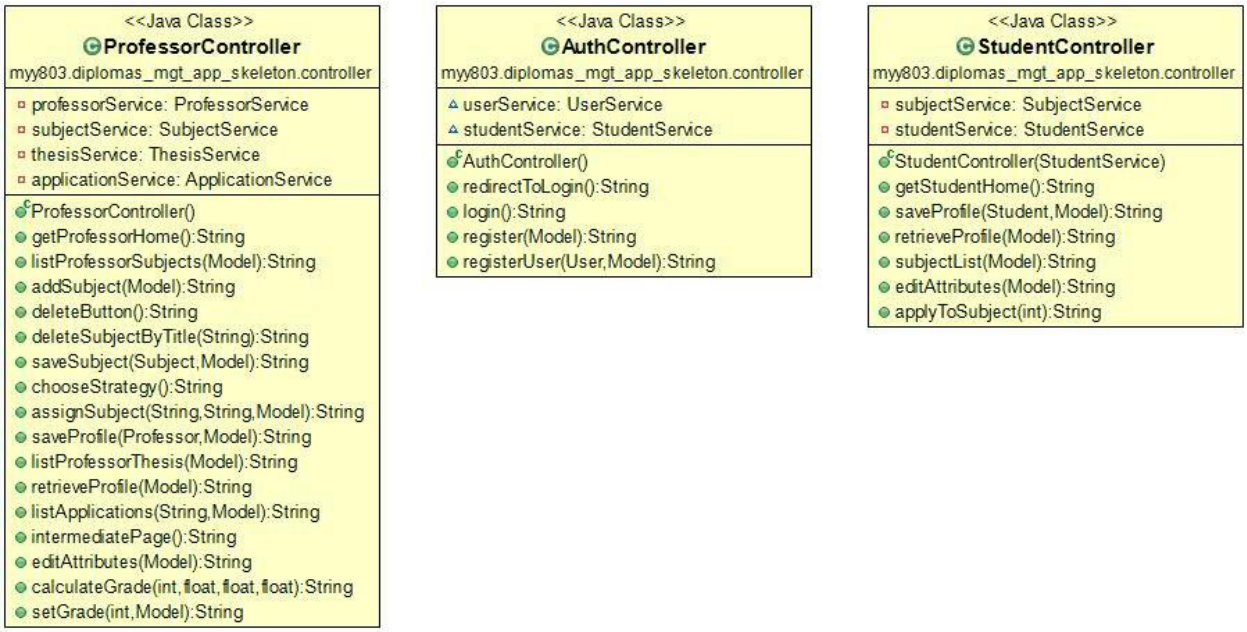
operations. ▪ Retrieve and save professor information. ▪ Assign subjects to students, based on strategies. ▪ Calculate and set grades for theses.	related to subjects such as retrieving subjects, assigning subjects to professors, and managing subject information. ▪ ThesisService: Collaborates with ThesisService to perform operations related to theses such as retrieving theses, setting grades for theses, and saving thesis information. ▪ ApplicationService: Collaborates with ApplicationService to perform operations related to applications such as retrieving applications and listing applications for a specific professor and subject. ▪ Model classes (Subject, Thesis, Application, Professor): Uses model classes to represent subjects, theses, applications, and professor information. ▪ BestApplicantStrategyFactory: Collaborates with BestApplicantStrategyFactory to create the appropriate strategy for selecting the best applicant for a subject. ▪ Authentication and SecurityContextHolder: Collaborates with authentication and security-related components to handle authentication and retrieve the current principal (professor). ▪ Spring Framework: Utilizes various Spring annotations and features for handling requests, dependency injection, and rendering views. ▪ View templates (professor/main-menu, professor/mySubjectList, professor/subject-form, professor/delete-subject-form, professor/prof-error, professor/strategy-form, professor/professor-form, professor/myThesesList,

	professor/setSubjectTitle, professor/myApplicationList, professor/setGrade): Renders the appropriate view templates to display the professor's main menu, subject list, subject form, delete subject form, error messages, strategy form, professor form, thesis list, set subject title form, application list, and set grade form.

Class Name: StudentController	
Responsibilities: ▪ Handle student-related functionalities and interactions. ▪ Manage student's profile, subjects, and applications. ▪ Render appropriate views and handle requests for student-related operations. ▪ Retrieve and save student information. ▪ List available subjects and allow students to apply to subjects. ▪ Provide a main menu for students.	**Collaborations:** ▪ SubjectService: Collaborates with SubjectService to perform operations related to subjects such as retrieving subjects and managing subject information. ▪ StudentService: Collaborates with StudentService to perform operations related to students such as retrieving student information, saving profiles, retrieving profiles, and managing student subjects and applications. ▪ Model classes (Student, Subject): Uses model classes to represent student and subject information. ▪ Authentication and SecurityContextHolder: Collaborates with authentication and security-related components to handle authentication and retrieve the current principal (student). ▪ Spring Framework: Utilizes various Spring annotations and features for handling requests, dependency injection, and rendering views. ▪ View templates (student/main-menu, student/list-allSubject,

	student/student-form): Renders the appropriate view templates to display the student's main menu, list of available subjects, and student form for editing student attributes.

UML:



CONFIG PACKAGE

Class Name: CustomSecuritySuccessHandler	
Responsibilities: ▪ Redirect the user to the appropriate URL after successful authentication. ▪ Determine the target URL based on the user's role (STUDENT or PROFESSOR). ▪ Handle HTTP requests and responses.	**Collaborations:** ▪ HttpServletRequest: Represents an HTTP request and is used to retrieve information about the request. ▪ HttpServletResponse: Represents an HTTP response and is used to send the response back to the client. ▪ Authentication: Represents the authenticated principal (user) after successful authentication. ▪ GrantedAuthority: Represents an authority granted to the authenticated principal.

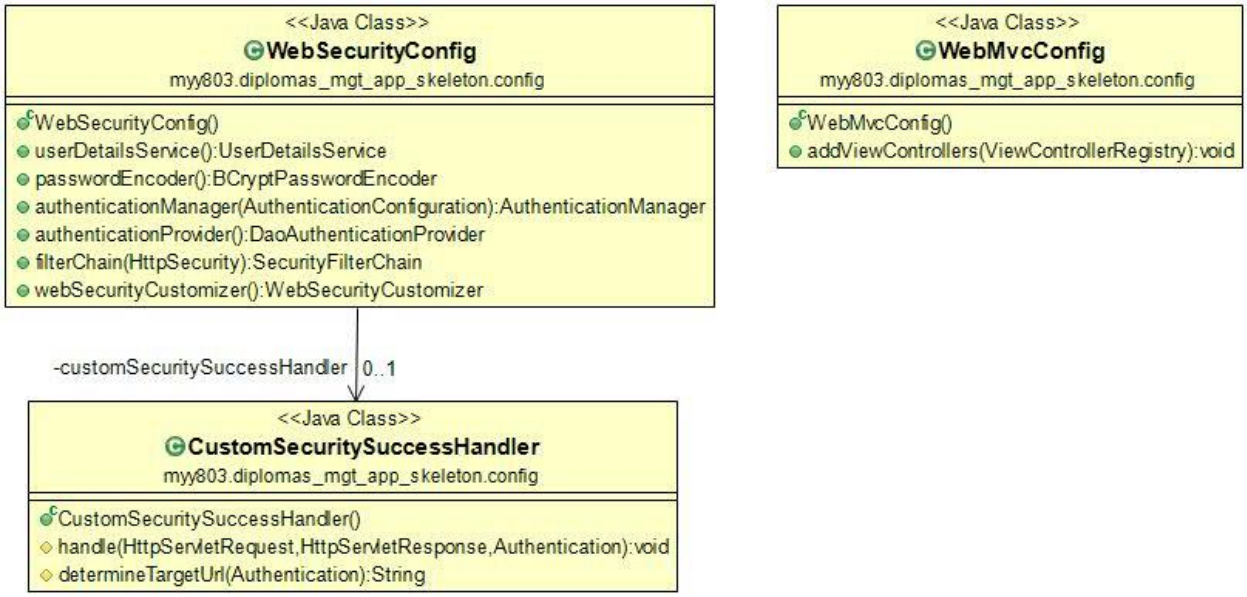
	▪ RedirectStrategy: Handles the redirection of the user to the target URL. ▪ DefaultRedirectStrategy: Default implementation of RedirectStrategy used for redirection. ▪ SimpleUrlAuthenticationSuccessHandler: A default implementation of an authentication success handler provided by Spring Security. ▪ Collection: Represents a generic collection of objects and is used to store GrantedAuthority objects. ▪ List: Represents an ordered collection of objects and is used to store the user's roles (authorities). ▪ Configuration: Indicates that the class is a configuration class and can be used by the Spring IoC container for bean definitions and other configurations.

Class Name: WebMvcConfig	
Responsibilities: ▪ Define view controllers for specific URLs without the need for a corresponding controller class.	**Collaborations:** ▪ ViewControllerRegistry: A registry for configuring simple automated controllers (view controllers) pre-configured with status code and/or a view. ▪ Configuration: Indicates that the class is a configuration class and can be used by the Spring IoC container for bean definitions and other configurations. ▪ WebMvcConfigurer: Interface to be implemented by types that register additional handlers and interceptors for the Spring MVC framework.

Class Name: WebSecurityConfig	
Responsibilities: ▪ Configure authentication and authorization rules for different URLs. ▪ Define the user details service for authentication. ▪ Define the password encoder for password hashing. ▪ Configure the authentication manager for handling authentication requests. ▪ Define the authentication provider for retrieving user details and verifying passwords. ▪ Configure the security filter chain.	**Collaborations:** ▪ Configuration: Indicates that the class is a configuration class and can be used by the Spring IoC container for bean definitions and other configurations. ▪ EnableWebSecurity: Enables Spring Security's web security support and provides the Spring MVC integration. ▪ CustomSecuritySuccessHandler: A custom implementation of the authentication success handler to determine the target URL after successful authentication. ▪ UserDetailsService: Interface for implementing a custom user details service. ▪ BCryptPasswordEncoder: Implementation of PasswordEncoder that uses the BCrypt strong hashing function to hash passwords. ▪ AuthenticationManager: Interface for managing authentication. ▪ DaoAuthenticationProvider: Implementation of AuthenticationProvider that retrieves user details from a UserDetailsService and authenticates a UsernamePasswordAuthenticationToken. ▪ HttpSecurity: Configures web-based security at the HTTP level. ▪ SecurityFilterChain: Represents the security filter chain used by Spring Security to apply security to HTTP requests. ▪ AntPathRequestMatcher: Matches URLs using the Ant-style path patterns. ▪ WebSecurityCustomizer: Interface for

	customizing WebSecurity configuration.

UML:



SKELETON PACKAGE

Class Name:	
Responsibilities: ▪ The main class serves as the entry point for executing the application.	**Collaborations:** -

UML: