DiplomasManagementApp

Sprint Report

Ομάδα 4426-4573-4482

Σπυρίδων Μοτσενίγος – 4426

Βασίλειος Μαρούλης – 4573

Σπυρίδων Πρίσκας - 4482

VERSIONS HISTORY

Date	Version	Description	Author
17/03/23	0.1	Setting up database and eclipse workspace.	4426-4573-4482
18/03/23	0.2	Making the 1st version of our backend.	4426-4573-4482
22/03/23	1.0	Adding log in/sign in functionality for users.	4426-4573-4482
24/03/23	1.1	Setting up our student , professor backend.	4426-4573-4482
27/03/23	1.2.1	Setting up our student and professor frontend.	4426-4573-4482
29/03/23	1.2.2	Adding personal information functionality for students and professor.	4426-4573-4482
02/04/23	2.0	Setting up our subject backend.	4426-4573-4482
06/04/23	2.1	Adding add/delete subject, functionality for professors.	4426-4573-4482
08/04/23	2.2	Extending add subject functionality (professor can now place more information)	4426-4573-4482
12/04/23	3.1	Adding submit/details functionalities for students adding UI elements for it.	4426-4573-4482
15/04/23	3.2	Refining the UI.	4426-4573-4482
18/04/23	4.0	Setting up our thesis and application backend.	4426-4573-4482
22/04/23	4.1	Setting up our application frontend.	4426-4573-4482
24/04/23	4.2	Setting up our strategies.	4426-4573-4482
28/04/23	4.3	Setting up final grade calculation and adding UI elements for it.	4426-4573-4482
02/05/23	4.5	Testing Implementation	4426-4573-4482
08/05/23	5.0	Fixing bugs – Final Version	4426-4573-4482

1 Introduction

1.1 Purpose

Develop a Web application that allows students to browse available diploma thesis projects from various professors and apply for the diploma thesis projects that interest them. The application further allows professors to assign diploma thesis projects to students, supervise the assigned theses projects and assess the outcomes.

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

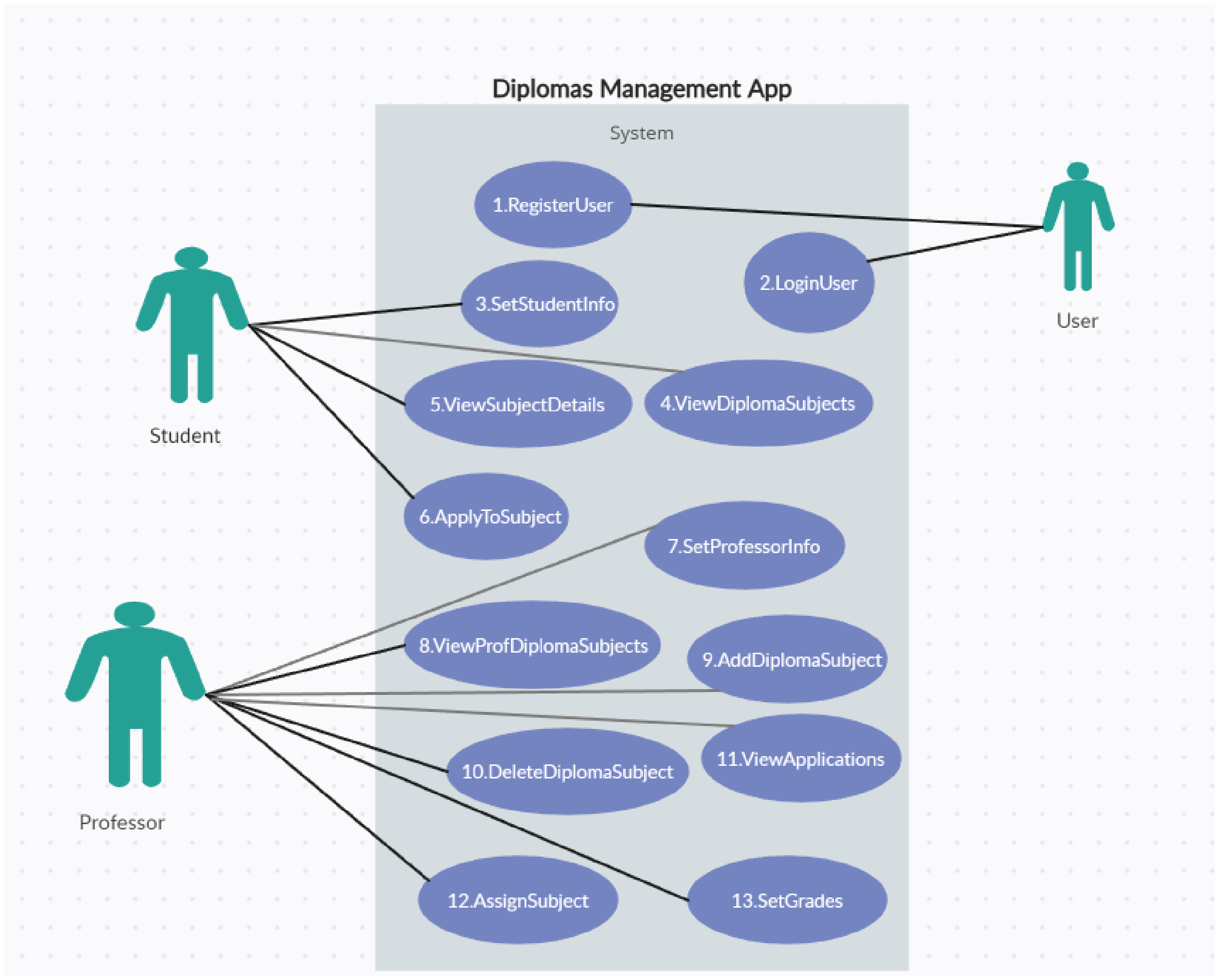
2.1 Scrum team

Product Owner	4426-4573-4482
Scrum Master	4426-4573-4482
Development Team	4426-4573-4482

2.2 Sprints

Sprint No	Begin Date	End Date	Number of weeks	User stories
1	**17/03/23**	**31/03/23**	**2**	**U1,U2,S1,P1**
2	**01/04/23**	**07/04/23**	**1**	**P2,P3,P4**
3	**08/04/23**	**14/04/23**	**1**	**S2,S3,S4**
4	**14/04/23**	**28/04/23**	**2**	**P5,P6,P7,P8,P9**
5	**28/04/23**	**05/05/23**	**1**	**Testing**

3 Use Cases


Diplomas Management App
System
1.RegisterUser
2.LoginUser
User
Student
3.SetStudentInfo
4.ViewDiplomaSubjects
5.ViewSubjectDetails
6.ApplyToSubject
7.SetProfessorInfo
8.ViewProfDiplomaSubjects
9.AddDiplomaSubject
Professor
10.DeleteDiplomaSubject
11.ViewApplications
12.AssignSubject
13.SetGrades

3.1 RegisterUser

Use case ID	UC1
Actors	User
Pre conditions	The user is not already registered
Main flow of events	1. The use case begins when the user clicks the register button. 2. The user provides their credentials. 3. The user selects their role (Student or professor) 4. The user submits the form. 5. The system creates a new account and stores the user information.
Post conditions	The user is registered to the application.

3.2 LoginUser

Use case ID	UC2
Actors	User
Pre conditions	The user has already registered to the application.
Main flow of events	1. The use case begins when the user clicks the login button. 2. The user provides their login username and password 3. The user submits the form. 4. The system checks that the login credentials are correct and the user exists. 5. The system grants the user access to the application.
Alternative flow 1	If the username or password is incorrect, the system informs the user
Post conditions	The user can operate the application.

3.3 SetStudentInfo

Use case ID	UC3
Actors	Student
Pre conditions	Student must have successfully logged in.
Main flow of events	1. The use case begins when the student clicks the Personal Info button. 2. The student fills in their full name, year of studies, average grade and number of remaining courses. 3. The student clicks the save button. 4. The system stores the student's info.
Post conditions	The student has set their personal info so that professors can evaluate them.

3.4 ViewDiplomaSubjects

Use case ID	UC4
Actors	Student
Pre conditions	Student must have successfully logged in.
Main flow of events	1. The use case begins when the student clicks the Available Subjects button. 2. The system displays a list of available subjects. 3. The student can view all the available diploma subjects.
Post conditions	The student can apply to any subjects that look interesting.

3.5 ViewSubjectDetails

Use case ID	UC5
Actors	Student
Pre conditions	Student must have successfully logged in.
Main flow of events	1. The use case begins when the student clicks the Details button on a specific subject. 2. The system displays a detailed description of the selected subject. 3. The student can view the diploma subject title and objectives.
Post conditions	The student can choose if they will actually apply to the subject.

3.6 ApplyToSubject

Use case ID	UC6
Actors	Student
Pre conditions	Student must have successfully logged in.
Main flow of events	1. The use case begins when the student clicks the Apply button on a specific subject. 2. The system creates a diploma thesis application for the student. 3. The system sends the application to the corresponding professor.
Post conditions	The professor know that the student is willing to take over the diploma thesis subject.

3.7 SetProfessorInfo

Use case ID	UC7
Actors	Professor
Pre conditions	Professor must have successfully logged in.
Main flow of events	1. The use case begins when the professor clicks the Personal Info button. 2. The professor fills in their full name and specialty. 3. The professor clicks the save button. 4. The system stores the professor's info.
Post conditions	The professor has set their personal info so that the students can see.

3.8 ViewProfDiplomaSubjects

Use case ID	UC8
Actors	Professor
Pre conditions	Professor must have successfully logged in.
Main flow of events	1. The use case begins when the professor clicks the Diploma Subjects button. 2. The system displays a list of all the diploma subjects the professor has created. 3. The professor can see the details of each subject.
Post conditions	The professor can view all the names and objectives of the diploma thesis subjects they have created.

3.9 AddDiplomaSubject

Use case ID	UC9
Actors	Professor
Pre conditions	Professor must have successfully logged in and browsed to the Diploma Subjects List page.
Main flow of events	1. The use case begins when the professor clicks the Add Diploma Subject button. 2. The professor fills in the title and objectives of the subject. 3. The professor saves the new subject. 4. The system stores the subject info and adds it to the diploma subject list.
Post conditions	The new diploma thesis subject is added to the subject list of the professor and available for the students to view and apply.

3.10 DeleteDiplomaSubject

Use case ID	UC10
Actors	Professor
Pre conditions	Professor must have successfully logged in and browsed to the Diploma Subjects List page.
Main flow of events	1. The use case begins when the professor clicks the Delete button on a selected subject. 2. The system asks the professor if they are sure about deleting the subject. 2.1. If the professor clicks yes, the system deletes the subject. 2.2. If the professor clicks cancel, nothing happens.
Alternative flow	If there are any applicants to the selected subject, the system does not delete it.
Post conditions	The selected subject is deleted and the subject list is up to date.

3.11 ViewApplications

Use case ID	UC11
Actors	Professor
Pre conditions	Professor must have successfully logged in and browsed to the Diploma Subjects List page.
Main flow of events	1. The use case begins when the professor clicks the Applicants button on a selected subject. 2. The system displays a list of applications from the students who want to take over the selected diploma thesis subject
Post conditions	The professor can view all the applications and can assign the diploma thesis subject to a particular student.

3.12 AssignSubject

Use case ID	UC12
Actors	Professor
Pre conditions	1. Professor must have successfully logged in and browsed to the Diploma Subjects List page. 2. Professor has created at least one diploma subject.
Main flow of events	1. The use case begins when the professor clicks the Applicants button on a selected subject. 2. The system displays a list of applications from the students who want to take over the selected diploma thesis subject. 3. The professor clicks the dropdown Automatically Assign Subject button. 3.1. If the professor clicks the best average grade button. 3.1.1. The system automatically finds the student application with the best average grade and assigns the subject to them 3.2. If the professor clicks the fewest remaining courses button. 3.2.1. The system automatically finds the student application with the

	fewest remaining courses and assigns the subject to them. 3.3. If the professor clicks the random choice button. 3.3.1. The system randomly selects an applicant assigns the subject to them.
Alternative flow 1	3. If the professor clicks the Assign button on a specific application: 3.1. The system automatically assigns the diploma thesis subject to that student.
Alternative flow 2	3. The professor fills in thresholds. 4. The professor clicks the filter button. 5. The system filters the applications based on the given thresholds. 6. The system displays only the applications that meet the criteria.
Post conditions	The selected student is assigned to the diploma thesis subject and can start working on the project.

3.13 ViewDiplomaTheses

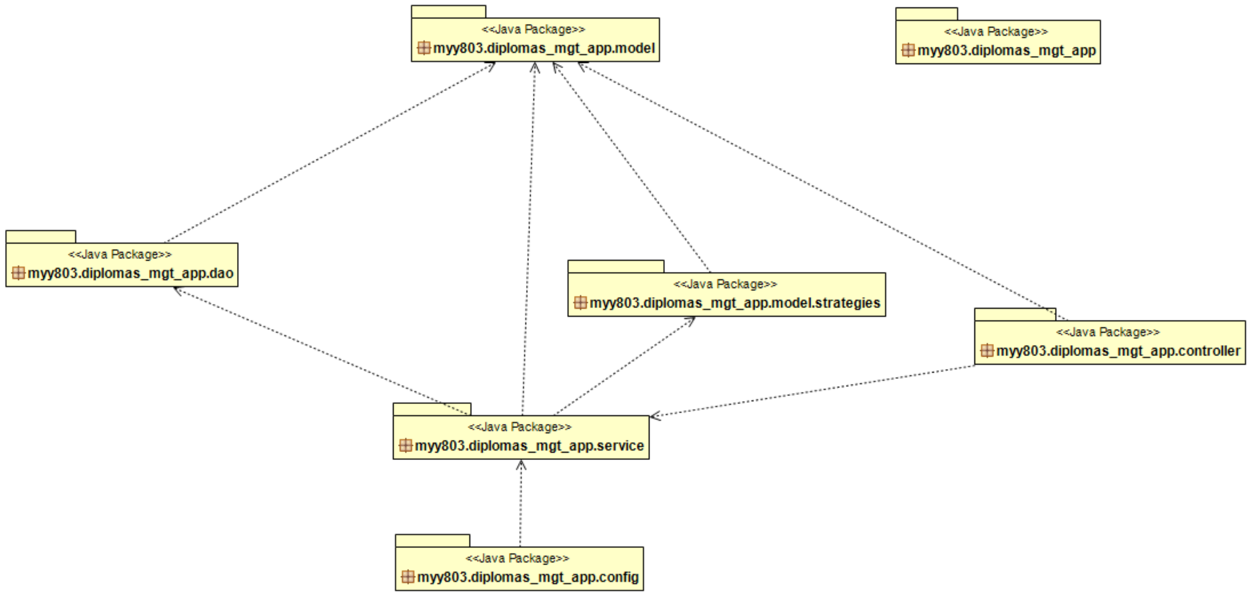
Use case ID	UC13
Actors	Professor
Pre conditions	1. Professor must have successfully logged in. 2. Professor has assigned at least one diploma thesis.
Main flow of events	1. The use case begins when the professor clicks the Theses button on the homepage. 2. The system displays a list of theses. 3. The professor views the student's name, subject and grades of each subject.
Post conditions	The professor can view a list of diploma theses and manage their related information

3.14 SetGrades

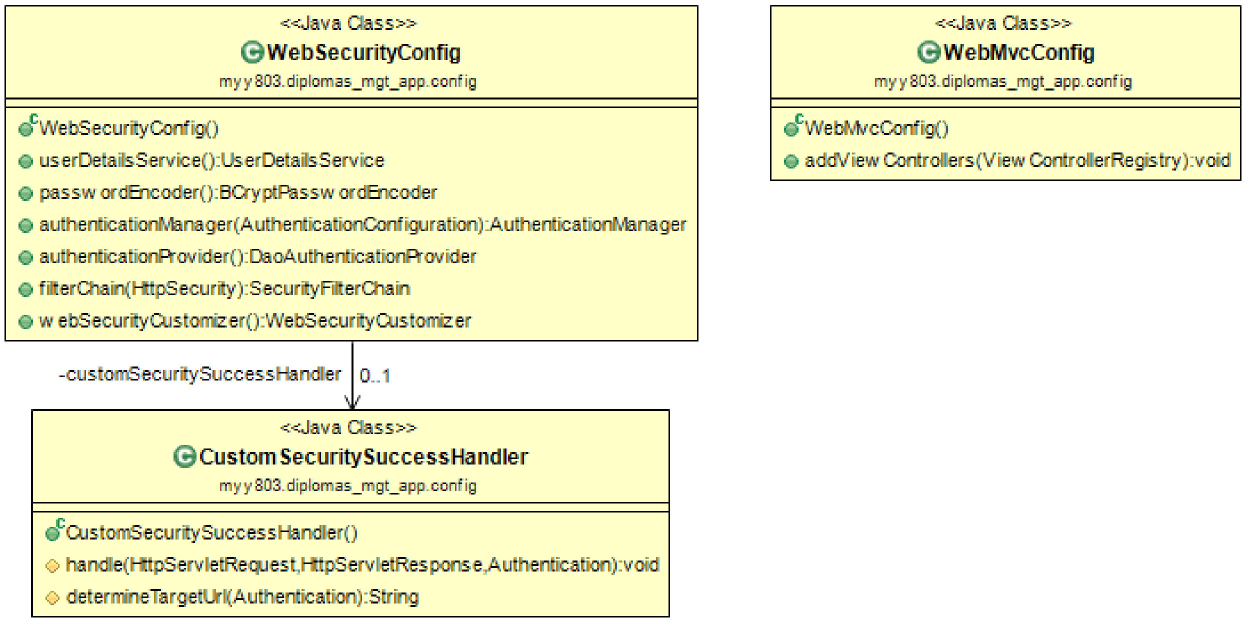
Use case ID	UC14
Actors	Professor
Pre conditions	1. Professor must have successfully logged in and browsed to the Theses page. 2. Professor has assigned at least one diploma thesis.
Main flow of events	1. The use case begins when the professor clicks the Grades button on a specific subject. 2. The system displays a page with the grades form. 3. The professor provides the implementation, report and presentation grade. 4. The professor clicks the Save / Calculate total grade button. 5. The system calculates the total grade based on the formula and saves the grades. 6. The system redirects back to the theses list page, where the professor can see the updated grades of the subject
Alternative flow	If the professor clicks the Back button, the system redirects back to the theses list page.
Post conditions	The grades for the implementation, report, and presentation are set for the diploma thesis and the total grade is calculated.

4 Design

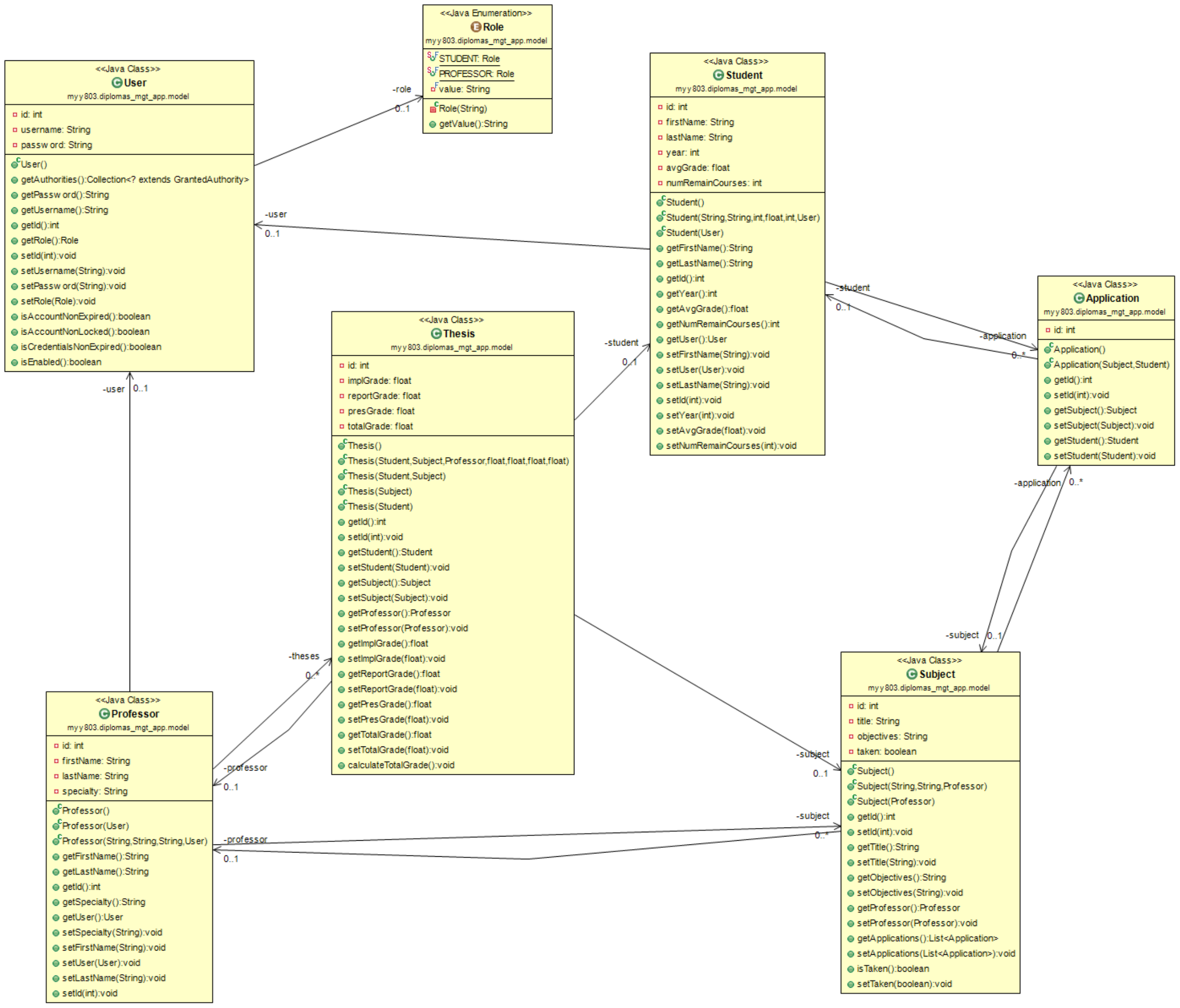
4.1 Architecture


<<Java Package>>
myy803.diplomas_mgt_app.model
<<Java Package>>
myy803.diplomas_mgt_app
<<Java Package>>
myy803.diplomas_mgt_app.dao
<<Java Package>>
myy803.diplomas_mgt_app.model.strategies
<<Java Package>>
myy803.diplomas_mgt_app.controller
<<Java Package>>
myy803.diplomas_mgt_app.service
<<Java Package>>
myy803.diplomas_mgt_app.config

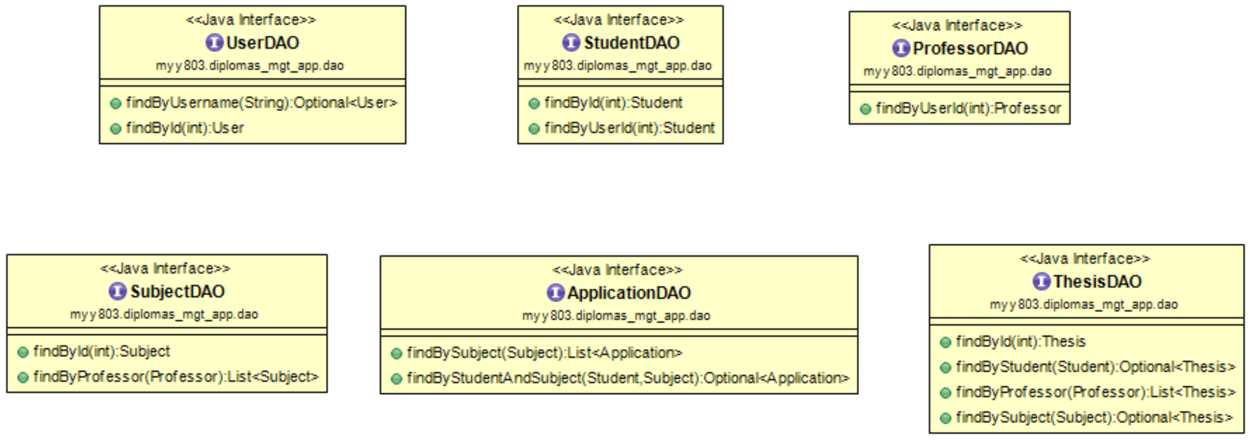
4.2 Design



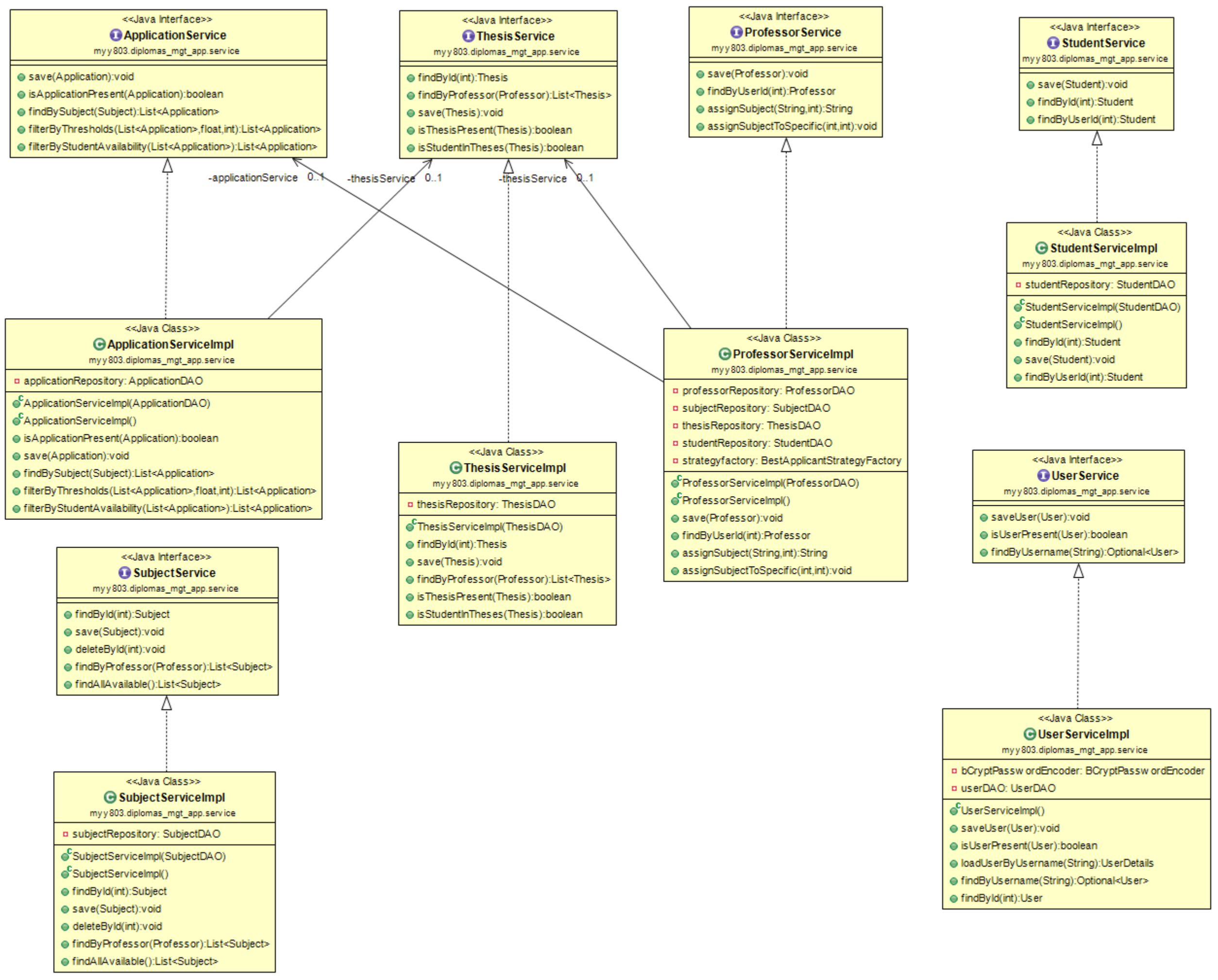
<<Java Class>>
WebSecurityConfig
myy803.diplomas_mgt_app.config
WebSecurityConfig()
userDetailsService():UserDetailsService
passwordEncoder():BCryptPasswordEncoder
authenticationManager(AuthenticationConfiguration):AuthenticationManager
authenticationProvider():DaoAuthenticationProvider
filterChain(HttpSecurity):SecurityFilterChain
webSecurityCustomizer():WebSecurityCustomizer
-customSecuritySuccessHandler
0..1
<<Java Class>>
CustomSecuritySuccessHandler
myy803.diplomas_mgt_app.config
CustomSecuritySuccessHandler()
handle(HttpServletRequest,HttpServletResponse,Authentication):void
determineTargetUrl(Authentication):String
<<Java Class>>
WebMvcConfig
myy803.diplomas_mgt_app.config
WebMvcConfig()
addViewControllers(ViewControllerRegistry):void

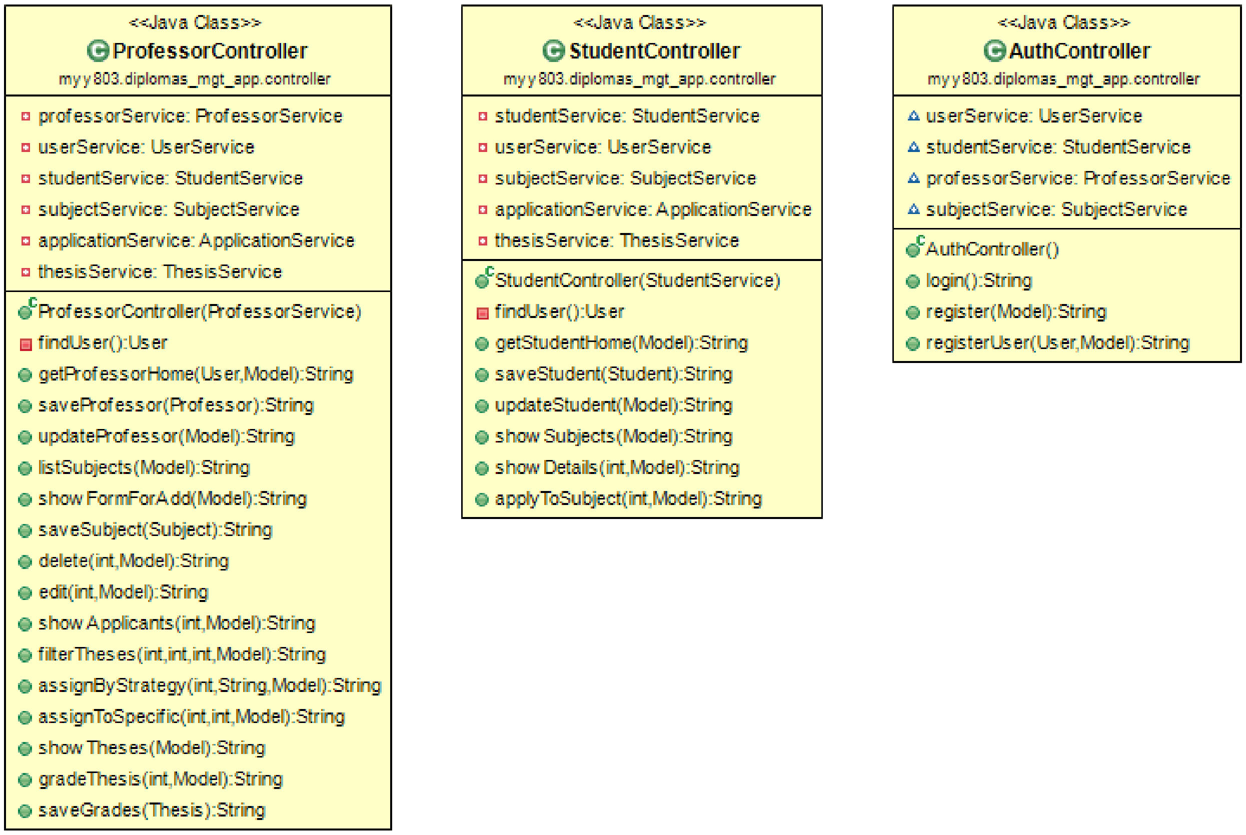


<<Java Enumeration>>
Role
myy803.diplomas_mgt_app.model
STUDENT: Role
PROFESSOR: Role
value: String
Role(String)
getValue():String
<<Java Class>>
User
myy803.diplomas_mgt_app.model
id: int
username: String
passw ord: String
User()
getAuthorities():Collection<? extends GrantedAuthority>
getPassw ord():String
getUsername():String
getId():int
getRole():Role
setId(int):void
setUsername(String):void
setPassw ord(String):void
setRole(Role):void
isAccountNonExpired():boolean
isAccountNonLocked():boolean
isCredentialsNonExpired():boolean
isEnabled():boolean
-role
0..1
-user
0..1
<<Java Class>>
Student
myy803.diplomas_mgt_app.model
id: int
firstName: String
lastName: String
year: int
avgGrade: float
numRemainCourses: int
Student()
Student(String,String,int,float,int,User)
Student(User)
getFirstName():String
getLastName():String
getId():int
getYear():int
getAvgGrade():float
getNumRemainCourses():int
getUser():User
setFirstName(String):void
setUser(User):void
setLastName(String):void
setId(int):void
setYear(int):void
setAvgGrade(float):void
setNumRemainCourses(int):void
-student
0..1
-application
0..*
<<Java Class>>
Application
myy803.diplomas_mgt_app.model
id: int
Application()
Application(Subject,Student)
getId():int
setId(int):void
getSubject():Subject
setSubject(Subject):void
getStudent():Student
setStudent(Student):void
-application
0..*
-subject
0..1
<<Java Class>>
Thesis
myy803.diplomas_mgt_app.model
id: int
implGrade: float
reportGrade: float
presGrade: float
totalGrade: float
Thesis()
Thesis(Student,Subject,Professor,float,float,float,float)
Thesis(Student,Subject)
Thesis(Subject)
Thesis(Student)
getId():int
setId(int):void
getStudent():Student
setStudent(Student):void
getSubject():Subject
setSubject(Subject):void
getProfessor():Professor
setProfessor(Professor):void
getImplGrade():float
setImplGrade(float):void
getReportGrade():float
setReportGrade(float):void
getPresGrade():float
setPresGrade(float):void
getTotalGrade():float
setTotalGrade(float):void
calculateTotalGrade():void
-student
0..1
-theses
0..*
-professor
0..1
-user
0..1
<<Java Class>>
Professor
myy803.diplomas_mgt_app.model
id: int
firstName: String
lastName: String
specialty: String
Professor()
Professor(User)
Professor(String,String,String,User)
getFirstName():String
getLastName():String
getId():int
getSpecialty():String
getUser():User
setSpecialty(String):void
setFirstName(String):void
setUser(User):void
setLastName(String):void
setId(int):void
-professor
0..1
-subject
0..1
-subject
0..*
<<Java Class>>
Subject
myy803.diplomas_mgt_app.model
id: int
title: String
objectives: String
taken: boolean
Subject()
Subject(String,String,Professor)
Subject(Professor)
getId():int
setId(int):void
getTitle():String
setTitle(String):void
getObjectives():String
setObjectives(String):void
getProfessor():Professor
setProfessor(Professor):void
getApplications():List<Application>
setApplications(List<Application>):void
isTaken():boolean
setTaken(boolean):void



<<Java Interface>>
UserDAO
myy803.diplomas_mgt_app.dao
findByUsername(String):Optional<User>
findById(int):User
<<Java Interface>>
StudentDAO
myy803.diplomas_mgt_app.dao
findById(int):Student
findByUserId(int):Student
<<Java Interface>>
ProfessorDAO
myy803.diplomas_mgt_app.dao
findByUserId(int):Professor
<<Java Interface>>
SubjectDAO
myy803.diplomas_mgt_app.dao
findById(int):Subject
findByProfessor(Professor):List<Subject>
<<Java Interface>>
ApplicationDAO
myy803.diplomas_mgt_app.dao
findBySubject(Subject):List<Application>
findByStudentAndSubject(Student,Subject):Optional<Application>
<<Java Interface>>
ThesisDAO
myy803.diplomas_mgt_app.dao
findById(int):Thesis
findByStudent(Student):Optional<Thesis>
findByProfessor(Professor):List<Thesis>
findBySubject(Subject):Optional<Thesis>


<<Java Interface>>
ApplicationService
myy803.diplomas_mgt_app.service
save(Application):void
isApplicationPresent(Application):boolean
findBySubject(Subject):List<Application>
filterByThresholds(List<Application>,float,int):List<Application>
filterByStudentAvailability(List<Application>):List<Application>
<<Java Interface>>
ThesisService
myy803.diplomas_mgt_app.service
findById(int):Thesis
findByProfessor(Professor):List<Thesis>
save(Thesis):void
isThesisPresent(Thesis):boolean
isStudentInTheses(Thesis):boolean
<<Java Interface>>
ProfessorService
myy803.diplomas_mgt_app.service
save(Professor):void
findByUserId(int):Professor
assignSubject(String,int):String
assignSubjectToSpecific(int,int):void
<<Java Interface>>
StudentService
myy803.diplomas_mgt_app.service
save(Student):void
findById(int):Student
findByUserId(int):Student
-applicationService 0..1
-thesisService 0..1
-thesisService 0..1
<<Java Class>>
StudentServiceImpl
myy803.diplomas_mgt_app.service
studentRepository: StudentDAO
StudentServiceImpl(StudentDAO)
StudentServiceImpl()
findById(int):Student
save(Student):void
findByUserId(int):Student
<<Java Class>>
ApplicationServiceImpl
myy803.diplomas_mgt_app.service
applicationRepository: ApplicationDAO
ApplicationServiceImpl(ApplicationDAO)
ApplicationServiceImpl()
isApplicationPresent(Application):boolean
save(Application):void
findBySubject(Subject):List<Application>
filterByThresholds(List<Application>,float,int):List<Application>
filterByStudentAvailability(List<Application>):List<Application>
<<Java Class>>
ProfessorServiceImpl
myy803.diplomas_mgt_app.service
professorRepository: ProfessorDAO
subjectRepository: SubjectDAO
thesisRepository: ThesisDAO
studentRepository: StudentDAO
strategyfactory: BestApplicantStrategyFactory
ProfessorServiceImpl(ProfessorDAO)
ProfessorServiceImpl()
save(Professor):void
findByUserId(int):Professor
assignSubject(String,int):String
assignSubjectToSpecific(int,int):void
<<Java Class>>
ThesisServiceImpl
myy803.diplomas_mgt_app.service
thesisRepository: ThesisDAO
ThesisServiceImpl(ThesisDAO)
findById(int):Thesis
save(Thesis):void
findByProfessor(Professor):List<Thesis>
isThesisPresent(Thesis):boolean
isStudentInTheses(Thesis):boolean
<<Java Interface>>
UserService
myy803.diplomas_mgt_app.service
saveUser(User):void
isUserPresent(User):boolean
findByUsername(String):Optional<User>
<<Java Interface>>
SubjectService
myy803.diplomas_mgt_app.service
findById(int):Subject
save(Subject):void
deleteById(int):void
findByProfessor(Professor):List<Subject>
findAllAvailable():List<Subject>
<<Java Class>>
UserServiceImpl
myy803.diplomas_mgt_app.service
bCryptPasswordEncoder: BCryptPasswordEncoder
userDAO: UserDAO
UserServiceImpl()
saveUser(User):void
isUserPresent(User):boolean
loadUserByUsername(String):UserDetails
findByUsername(String):Optional<User>
findById(int):User
<<Java Class>>
SubjectServiceImpl
myy803.diplomas_mgt_app.service
subjectRepository: SubjectDAO
SubjectServiceImpl(SubjectDAO)
SubjectServiceImpl()
findById(int):Subject
save(Subject):void
deleteById(int):void
findByProfessor(Professor):List<Subject>
findAllAvailable():List<Subject>


<<Java Class>>
ProfessorController
myy803.diplomas_mgt_app.controller
professorService: ProfessorService
userService: UserService
studentService: StudentService
subjectService: SubjectService
applicationService: ApplicationService
thesisService: ThesisService
ProfessorController(ProfessorService)
findUser():User
getProfessorHome(User,Model):String
saveProfessor(Professor):String
updateProfessor(Model):String
listSubjects(Model):String
showFormForAdd(Model):String
saveSubject(Subject):String
delete(int,Model):String
edit(int,Model):String
showApplicants(int,Model):String
filterTheses(int,int,int,Model):String
assignByStrategy(int,String,Model):String
assignToSpecific(int,int,Model):String
showTheses(Model):String
gradeThesis(int,Model):String
saveGrades(Thesis):String
<<Java Class>>
StudentController
myy803.diplomas_mgt_app.controller
studentService: StudentService
userService: UserService
subjectService: SubjectService
applicationService: ApplicationService
thesisService: ThesisService
StudentController(StudentService)
findUser():User
getStudentHome(Model):String
saveStudent(Student):String
updateStudent(Model):String
showSubjects(Model):String
showDetails(int,Model):String
applyToSubject(int,Model):String
<<Java Class>>
AuthController
myy803.diplomas_mgt_app.controller
userService: UserService
studentService: StudentService
professorService: ProfessorService
subjectService: SubjectService
AuthController()
login():String
register(Model):String
registerUser(User,Model):String

Class Name: CoursesMgtApp	
Responsibilities: ▪ Main class ▪ Starts the application	**Collaborations:** -

Class Name: CustomSecuritySuccessHandler	
Responsibilities: ▪ Secure authentication of users	**Collaborations:**

Class Name: WebSecurityConfig	
Responsibilities: ▪ Secure authentication of users	**Collaborations:** ▪ CustomSecuritySuccessHandler

Class Name: WebMvcConfig	
Responsibilities: ▪ Configures default url homepage	**Collaborations:** ▪ WebSecurityConfig

Class Name: AuthController	
Responsibilities: ▪ Controls the interface for user login and registration	**Collaborations:** ▪ User ▪ UserService ▪ Role

Class Name: StudentController	
Responsibilities: ▪ Controls the interface for the students	**Collaborations:** ▪ Student ▪ StudentService ▪ User

Class Name: ProfessorController	
Responsibilities: ▪ Controls the interface for the professors	**Collaborations:** ▪ Professor ▪ ProfessorService ▪ User

Class Name: Interface UserService	
Responsibilities: ▪ Controls the interface for the auth controller	**Collaborations:** ▪ User ▪ UserServiceImpl ▪ AuthController ▪ StudentController ▪ ProfessorController

Class Name: UserServiceImpl	
Responsibilities: ▪ Implements user service ▪ Business logic	**Collaborations:** ▪ User ▪ UserDAO ▪ UserService ▪ WebSecurityMvcConfig

Class Name: Interface StudentService	
Responsibilities: ▪ Controls the interface for the student controller	**Collaborations:** ▪ Student ▪ StudentServiceImpl ▪ AuthController ▪ StudentController ▪ ProfessorController

Class Name: StudentServiceImpl	
Responsibilities: ▪ Implements student service ▪ Business logic	**Collaborations:** ▪ Student ▪ StudentDAO ▪ StudentService

Class Name: Interface ProfessorService	
Responsibilities: ▪ Controls the interface for the professor controller	**Collaborations:** ▪ Professor ▪ ProfessorServiceImpl ▪ AuthController ▪ ProfessorController ▪ StudentController

Class Name: ProfessorServiceImpl	
Responsibilities: ▪ Implements professor service ▪ Business logic	**Collaborations:** ▪ Professor ▪ ProfessorDAO ▪ ProfessorService

Class Name: Interface SubjectService	
Responsibilities: ▪ Controls the interface for the controllers	**Collaborations:** ▪ Subject ▪ SubjectServiceImpl ▪ Professor ▪ ProfessorController ▪ StudentController

Class Name: SubjectServiceImpl	
Responsibilities: ▪ Implements subject service ▪ Business logic	**Collaborations:** ▪ Subject ▪ SubjectDAO ▪ SubjectService

Class Name: Interface ApplicationService	
Responsibilities: ▪ Controls the interface for the controllers	**Collaborations:** ▪ Application ▪ ApplicationServiceImpl ▪ ProfessorController ▪ StudentController

Class Name: ApplicationServiceImpl	
Responsibilities: ▪ Implements application service ▪ Business logic	**Collaborations:** ▪ Application ▪ ApplicationDAO ▪ ApplicationService

Class Name: Interface ThesisService	
Responsibilities: ▪ Controls the interface for the controllers	**Collaborations:** ▪ Thesis ▪ ThesisServiceImpl ▪ Professor ▪ ProfessorController ▪ StudentController

Class Name: ThesisServiceImpl	
Responsibilities: ▪ Implements thesis service ▪ Business logic	**Collaborations:** ▪ Thesis ▪ ThesisDAO ▪ ThesisService

Class Name: Interface UserDAO	
Responsibilities: ▪ Database queries ▪ Creates, updates and deletes fields	**Collaborations:** ▪ UserServiceImpl

Class Name: Interface StudentDAO	
Responsibilities: ▪ Database queries ▪ Creates, updates and deletes fields	**Collaborations:** ▪ StudentServiceImpl ▪ ProfessorServiceImpl

Class Name: Interface ProfessorDAO	
Responsibilities: ▪ Database queries ▪ Creates, updates and deletes fields	**Collaborations:** ▪ ProfessorServiceImpl

Class Name: Interface SubjectDAO	
Responsibilities: ▪ Database queries ▪ Creates, updates and deletes fields	**Collaborations:** ▪ SubjectServiceImpl ▪ ProfessorServiceImpl

Class Name: Interface ApplicationDAO	
Responsibilities: ▪ Database queries ▪ Creates, updates and deletes fields	**Collaborations:** ▪ ApplicationServiceImpl

Class Name: Interface ThesisDAO	
Responsibilities: ▪ Database queries ▪ Creates, updates and deletes fields	**Collaborations:** ▪ ThesisServiceImpl ▪ ProfessorServiceImpl

Class Name: User	
Responsibilities: ▪ Create and manage entries on the database table users ▪ Model Class	**Collaborations:** ▪ AuthController ▪ Student ▪ Professor ▪ StudentController ▪ ProfessorController

Class Name: Student	
Responsibilities: ▪ Create and manage entries on the database table students ▪ Model Class	**Collaborations:** ▪ User ▪ Application ▪ Thesis ▪ StudentController ▪ ProfessorController

Class Name: Professor	
Responsibilities: ▪ Create and manage entries on the database table professors ▪ Model Class	**Collaborations:** ▪ User ▪ Thesis ▪ Subject ▪ StudentController ▪ ProfessorController

Class Name: Subject	
Responsibilities: ▪ Create and manage entries on the database table subjects ▪ Model Class	**Collaborations:** ▪ Application ▪ Thesis ▪ Professor ▪ StudentController ▪ ProfessorController

Class Name: Application	
Responsibilities: ▪ Create and manage entries on the database table applications ▪ Model Class	**Collaborations:** ▪ Student ▪ Subject ▪ StudentController ▪ ProfessorController

Class Name: Thesis	
Responsibilities: ▪ Create and manage entries on the database table theses ▪ Model Class	**Collaborations:** ▪ Student ▪ Professor ▪ Subject ▪ StudentController ▪ ProfessorController

Class Name: Interface BestApplicantStrategy	
Responsibilities: ▪ Provides an interface for TemplateStrategyAlgorithm ▪ Provided a list of applications find the most suitable student based on strategy	**Collaborations:** ▪ TemplateStrategyAlgorithm ▪ BestApplicantStrategyFactory

Class Name: Abstract Class TemplateStrategyAlgorithm	
Responsibilities: ▪ Implements the interface	**Collaborations:** ▪ BestAvgGradeStrategy ▪ RandomChoiceStrategy ▪ FewestCoursesStrategy ▪ BestApplicantStrategy

Class Name: BestApplicantStrategyFactory	
Responsibilities: ▪ Creates strategy	**Collaborations:** ▪ BestAvgGradeStrategy ▪ RandomChoiceStrategy ▪ FewestCoursesStrategy ▪ BestApplicantStrategy

Class Name: BestAvgGradeStrategy	
Responsibilities: ▪ Selects the student with the best average grade	**Collaborations:** ▪ TemplateStrategyAlgorithm

Class Name: FewestCoursesStrategy	
Responsibilities: ▪ Selects the student with the fewest remaining courses	**Collaborations:** ▪ TemplateStrategyAlgorithm

Class Name: RandomChoiceStrategy	
Responsibilities: ▪ Selects a random student	**Collaborations:** ▪ TemplateStrategyAlgorithm