CoursesManagementApp

Sprint Report

Binas Basilis, 4434

Bintzilaiou Nikoletta, 4435

Mylonas Dionysis, 4443

VERSIONS HISTORY

Date	Version	Description	Author
18/5	1	Secure Web application	

1 Introduction

This document provides information concerning the final sprint of the project.

1.1 Purpose

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

TEST CASE ID	TEST SCENARIO	TEST CASE	PRE-CONDITION	TEST STEPS	EXPECTED RESULT	POST CONDITION
`testFindById` `ReturnsCourse` `(US 3)`	Linear algebra 1 has the correct id	Course with id 1 is Linear Algebra 1	Course with id 1 exists	-	Succesufully validate that the course with id 1 is linear algebra 1	-
`testFindByProf` `UsernameReturnsList` `(US 2)`	A user teaches the correct amount of courses	Zarras teaches 3 courses	Zarass and at least 3 courses exist	-	Validate that zarras teaches indeed 3 courses	-
`TestUpdateCourse` `(US 5)`	A course can be updated	Course with id 5 will have it's year updated	Course with id 5 exists	-	Update will hapenn successfully	Course with id 5 will have it's year as 5
`TestDeleteCourse` `(US 4)`	A course can be deleted	Course with id 4 will be deleted	Course with id 4 exists	-	Deletion will happen succesfuly	There is no longer a course with id 4
`testFindByRegId`	Niki is the correct class	Niki is in course with id 1	Niki and course with	-	Niki has been found in the	-

`ReturnsRegistration` `(US 6,7)`			id 1 exist		registration list of course with id 1	
`testFindAllBy` `RegId_CourseId` `(US 6)`	Checks the number of students in a course	There is 1 student in course with the id 1	A student and course with id 1 exist	-	Niki has been found in the course with id 1 registration list successfully	-
`testFindByRegId` `ReturnsGrade` `(US 10,11)`	A student has been graded a specific grade	Student with the id 4435 has been graded correctly	A student and a course exist	-	Student with the id 4435 has the grades 5,5 in course with the id 1	-
`testFindAllBy` `RegId_CourseId` `(US 10,11)`	All students in a course have been graded	Students of the course with id 1 have a grade	Course with the id 1 and a student that attends it exist	-	Successfully validate that all of the students in the course with id 1 have been given a grade	-
`testSaveCourse` `ReturnsPage` `(US 3)`	Save a new course	Ethics has been successfully created	-	-	Ethics is now a valid course with id 100	Add ethics in the table of courses

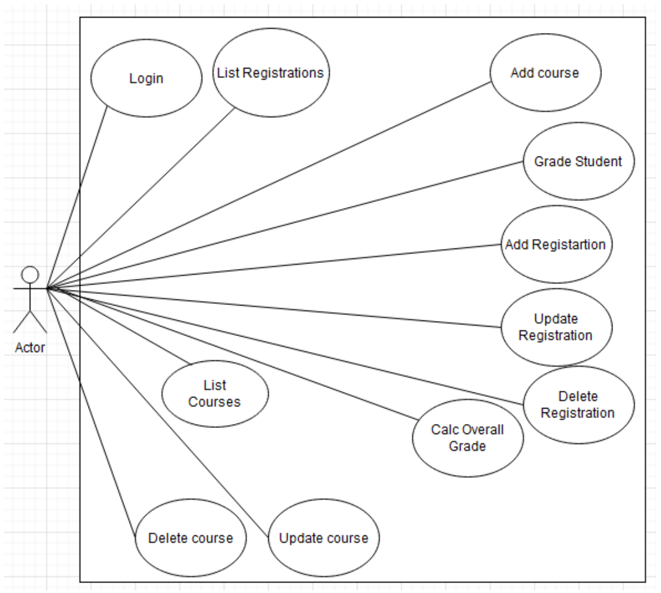
(All the service tests are similar in form to the dao tests so everything that applies to the dao tests also applies to the service tests and will thus be excluded)

2.1 Sprints

Sprint No	Begin Date	End Date	Number of weeks	User stories
0	12/3	15/3	0	Reading the requierments definition, strategize
0	16/3	20/3	0	Brainstorm ideas for the implementations, split work

0	21/3	27/3	1	-
1	28/3	4/4	1	US2, US3, US4
2	5/4	11/4	1	US5, US6
3	11/4	18/4	1	US7, US8 (VER1)
3	19/4	21/4	0	US7, US8 (VER2)
3	22/4	25/4	0	-
4	25/4	27/4	0	US9 (VER1)
4	28/4	1/5	0	US1 (STUCK)
5	6/5	9/5	0	US7, US8, US9 (FINAL VER)
6	9/5	10/5	0	US10, US11
7	10/5	12/5	0	US1
8	12/5	14/5	0	Test cases
9	13/5	16/7	0	Final changes to all US

3 Use Cases



3.1 <Use Case 1>

Use case ID	01
Actors	Instructor
Pre conditions	The user has run the application and opened a new bowser on localhost:8080
Main flow of events	1. The use case starts when the user enters their username and their password(same as the username) in the corresponding forms. 2. The user presser the Sign in button
Alternative flow 1	If the user enters incorrect username or password, then the error “Bad credential” is shown.
Alternative flow 2	If the user presser the Sign in button without providing a username and/or password, nothing happens
Post conditions	The user can now see the index page and click on the link to see his courses.

3.2 <Use Case 2>

Use case ID	02
Actors	Instructor
Pre conditions	The user has successfully logged in.
Main flow of events	1. The use case starts when the user clicks on the link in the index page to see the list of his courses.
Alternative flow 1	The user exits the application by pressing the Sign out button
Alternative flow 2	The user closes the browser.
Post conditions	The user is shown the list of his courses.

3.3 <Use Case 3>

Use case ID	03

Actors	Instructor
Pre conditions	The user has successfully logged in and the list of their courses is shown.
Main flow of events	1. The use case starts when the user presses the Add Course button. 2. The user enters the relevant information for the new course: course name, syllabus, year, semester, description etc. 3. The user presses the Save button.
Alternative flow 1	The user exits the application by pressing the logout button.
Alternative flow 2	If the user enters incorrect type of input in the form, f.e. characters in the year field, and then presses Save, an error page is shown.
Post conditions	The user is redirected to the list of his courses and can now see the new one added.

3.4 <Use Case 4>

Use case ID	04
Actors	Instructor
Pre conditions	The user has successfully logged in and the list of their courses is shown. The list has at least 1 course to delete.
Main flow of events	1. The use case starts when the user goes to the course they want to alter and then presses the Delete Course button that is in the action column. 2. A window pops up, asking “Are you sure you want to delete this course?” a. If the user presses Yes, the course is removed from the list. b. If the user presses Exit, the course remains in the list.
Alternative flow 1	The user exits the application by pressing the logout button.
Post conditions	The course list is updated and the course no longer exists.

3.5 <Use Case 5>

Use case ID	05
Actors	Instructor
Pre conditions	The user has successfully logged in and the list of their courses is shown. The list has at least 1 course to update.
Main flow of events	1. The use case starts when the user goes to the course they want to alter and then presses the Edit Course button that is in the Action column. 2. They fill in the form and then press Save.
Alternative flow 1	The user exits the application by pressing the logout button.
Alternative flow 2	The user presses the back to list link without changing anything and is redirected to the list of courses.
Post conditions	The user is redirected to the list of his courses and can now see that the specified course's info is updated.

3.6 <Use Case 6>

Use case ID	06
Actors	Instructor
Pre conditions	The user has successfully logged in and the list of their courses is shown. The list has at least 1 course.
Main flow of events	1. The use case starts when the user presses the Registrations button in the Action column, next to the course they are interested in.
Alternative flow 1	The user exits the application by pressing the logout button.
Post conditions	The user can now browse the list of students enrolled in that particular course.

3.7 <Use Case 7>

Use case ID	07
Actors	Instructor
Pre	The user has successfully logged in and the list of his courses is shown. The list

conditions	has at least 1 course.
Main flow of events	1. The use case starts when the user goes to the course they want to add the student to and then presses the Add Student button in the Action column. 2. The user enters the relevant information for the new student: student name, year of registration, semester and the presses the Save info button.
Alternative flow 1	The user exits the application by pressing the logout button.
Alternative flow 2	If the user enters incorrect type of input in the form, f.e. characters in the year of registration field, and then presses Save, an error page is shown.
Post conditions	The list of students is updated and the new student is now added to it.

3.8 <Use Case 8>

Use case ID	08
Actors	Instructor
Pre conditions	The user has successfully logged in and the list of the students that are enrolled in a specific course is shown. The list of registrations has at least 1 student.
Main flow of events	1. The use case starts when the user goes to the student they want to remove and then presses the Delete student button that is in the action column. 2. A window pops up, asking "Are you sure you want to delete this student?" c. If the user presses Yes, the student is removed from the list. d. If the user presses Exit, the student remains in the list
Alternative flow 1	The user exits the application by pressing the logout button
Post conditions	The list of students is updated and the selected student isn't in the list anymore. The user is redirected to the list of courses.

3.9 <Use Case 9>

Use case ID	09
Actors	Instructor
Pre conditions	The user has successfully logged in, the list of the students that are enrolled in a specific course is shown and it has at least one student in it.
Main flow of events	1. The use case starts when the user goes to the student whose information they want to change and then presses the Update info button that is in the action column. 2. They fill in the form and then press Save info.
Alternative flow 1	The user presses the back to list link without changing anything and is redirected to the list of courses.
Alternative flow 2	If the user enters incorrect type of input in the form, f.e. characters in the year of registration field, and then presses Save, an error page is shown.
Post conditions	The list of students is updated and the selected student's new info has replaced the old info. The user is redirected to the list of courses.

3.10 <Use Case 10>

Use case ID	10
Actors	Instructor
Pre conditions	The user has successfully logged in, the list of the students that are enrolled in a specific course is shown and it has at least one student in it.
Main flow of events	1. The use case starts when the user goes to the student who they want to grade and then presses the Grade student button that is in the action column. 2. They fill in the form for the final exam and the project grade, then press Save Grading. 3. If the user presses the Show Grades button at the top of the list of students page, the can see the grading an delete it, if needed.
Alternative flow 1	The user presses the back to list link without changing anything and is redirected to the list of courses.
Alternative flow 2	If the user enters incorrect type of input in the form, f.e. characters in the grades fields, and then presses Save, an error page is shown.
Post conditions	The grades of the student are registered. The user is redirected to the list of courses.

3.11 <Use Case 11>

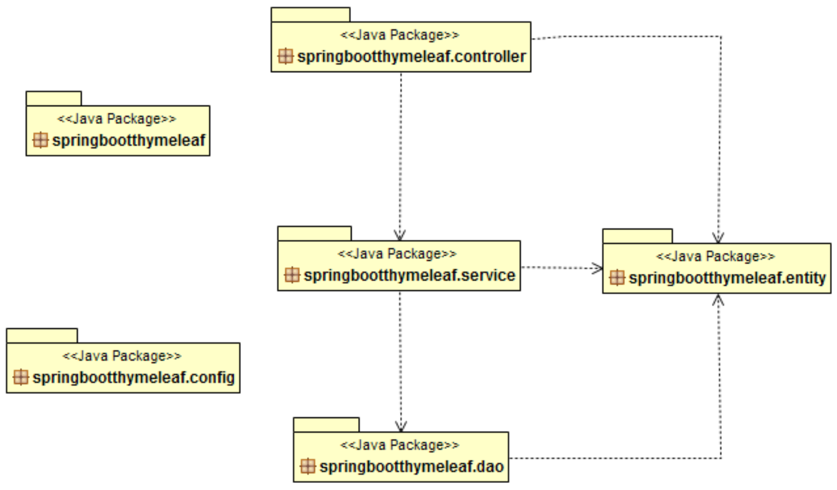
Use case ID	11
Actors	Instructor
Pre conditions	The user has successfully logged in, the list of the students that are enrolled in a specific course has at least one person in and they have been graded. The list of all the grades in the course is shown.
Main flow of events	1. The use case starts when the user presses the Overall Grade of course button.
Alternative flow 1	The user exits the application by closing the browser.
Alternative flow 2	The user presses the back to list link without changing anything and is redirected to the list of courses.
Post conditions	The overall grade of each student is shown. Also, the average grade of the students in the course is shown.

3.12 <Use Case 12>

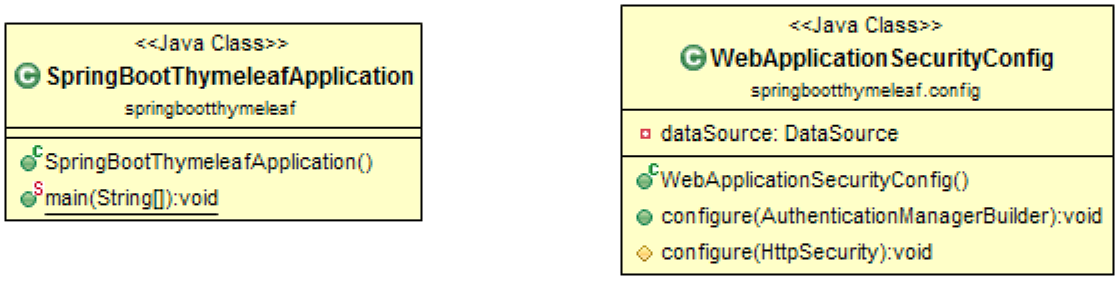
Not implemented

4 Design

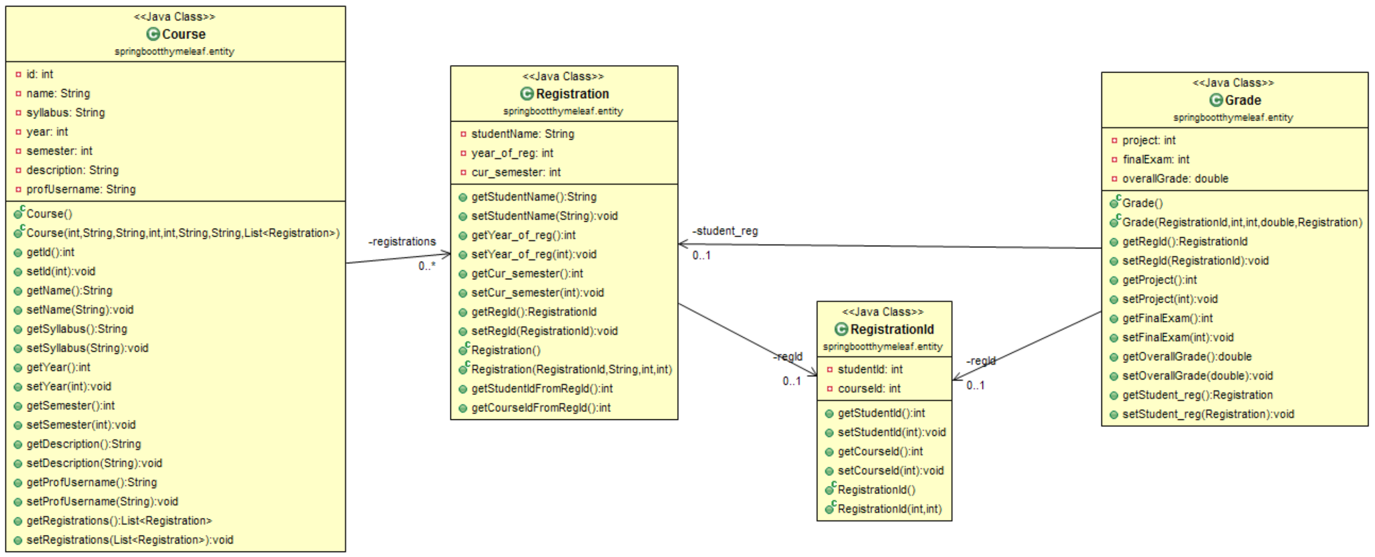
4.1 Architecture



4.2 Design



<<Java Class>>
SpringBootThymeleafApplication
springbootthymeleaf
SpringBootThymeleafApplication()
main(String[]):void
<<Java Class>>
WebApplicationSecurityConfig
springbootthymeleaf.config
dataSource: DataSource
WebApplicationSecurityConfig()
configure(AuthenticationManagerBuilder):void
configure(HttpSecurity):void



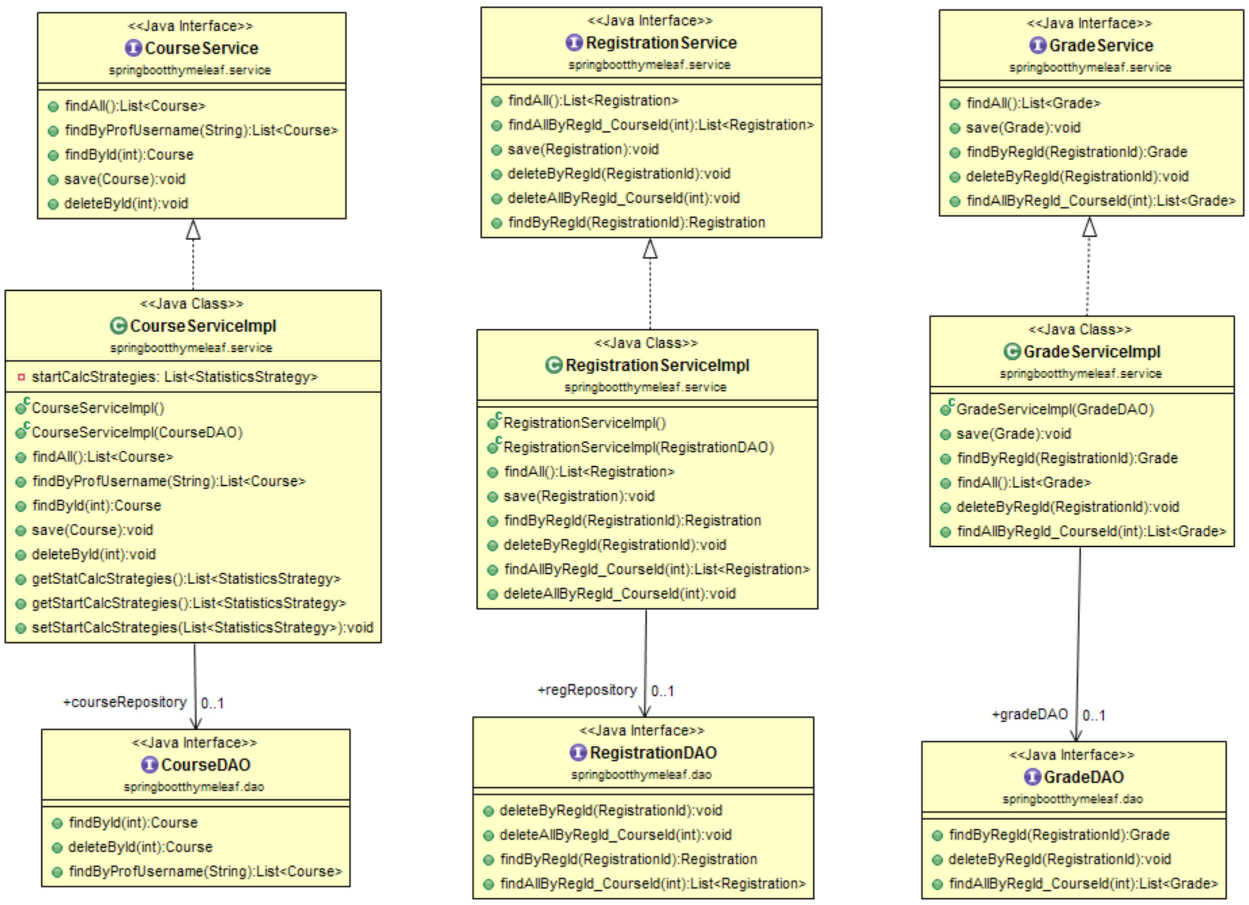
Class Name: Course	
Responsibilities:	**Collaborations:**
▪ Describes the JPA entity Course that corresponds to a row of the courses table and makes course objects.	▪ This class is used by the Courses DAO, to find/store courses in the database. ▪ The CourseMgtAppController uses this class to implement relevant user stories.

Class Name: Registration	
Responsibilities:	**Collaborations:**
▪ Describes the JPA entity Registration that corresponds to a row of the registrations table and makes registration objects.	▪ This class is used by the Registrations DAO, to find/store student registrations in the database. ▪ The CourseMgtAppController uses this

	class to implement relevant user stories.

Class Name: RegistrationId	
Responsibilities: ▪ Holds the embedded id of the entity Registration, consitsting of it's own student id and a course's unique id.	**Collaborations:** ▪ This class is used by the Registration class, a RegistrationId is part of a Registration object. ▪ The CourseMgtAppController uses this class to implement relevant user stories regarding students.

Class Name: Grade	
Responsibilities: ▪ Describes the JPA entity Grade that corresponds to a row of the grades table and makes grade objects.	**Collaborations:** ▪ The CourseMgtAppController uses this class to implement relevant user stories.

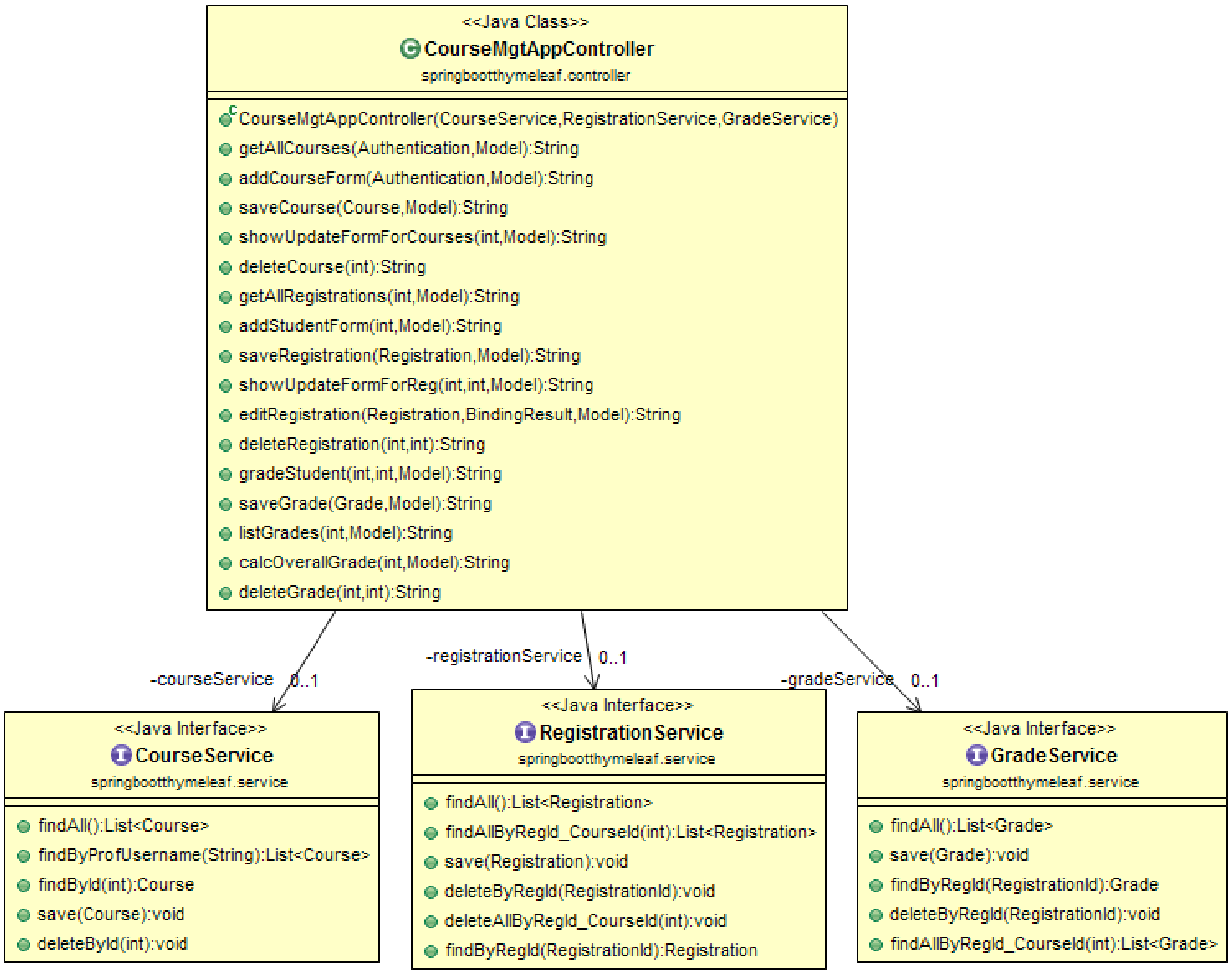


<<Java Interface>>
CourseService
springbootthymeleaf.service
findAll():List<Course>
findByProfUsername(String):List<Course>
findById(int):Course
save(Course):void
deleteById(int):void
<<Java Interface>>
RegistrationService
springbootthymeleaf.service
findAll():List<Registration>
findAllByRegId_CourseId(int):List<Registration>
save(Registration):void
deleteByRegId(RegistrationId):void
deleteAllByRegId_CourseId(int):void
findByRegId(RegistrationId):Registration
<<Java Interface>>
GradeService
springbootthymeleaf.service
findAll():List<Grade>
save(Grade):void
findByRegId(RegistrationId):Grade
deleteByRegId(RegistrationId):void
findAllByRegId_CourseId(int):List<Grade>
<<Java Class>>
CourseServiceImpl
springbootthymeleaf.service
startCalcStrategies: List<StatisticsStrategy>
CourseServiceImpl()
CourseServiceImpl(CourseDAO)
findAll():List<Course>
findByProfUsername(String):List<Course>
findById(int):Course
save(Course):void
deleteById(int):void
getStatCalcStrategies():List<StatisticsStrategy>
getStartCalcStrategies():List<StatisticsStrategy>
setStartCalcStrategies(List<StatisticsStrategy>):void
<<Java Class>>
RegistrationServiceImpl
springbootthymeleaf.service
RegistrationServiceImpl()
RegistrationServiceImpl(RegistrationDAO)
findAll():List<Registration>
save(Registration):void
findByRegId(RegistrationId):Registration
deleteByRegId(RegistrationId):void
findAllByRegId_CourseId(int):List<Registration>
deleteAllByRegId_CourseId(int):void
<<Java Class>>
GradeServiceImpl
springbootthymeleaf.service
GradeServiceImpl(GradeDAO)
save(Grade):void
findByRegId(RegistrationId):Grade
findAll():List<Grade>
deleteByRegId(RegistrationId):void
findAllByRegId_CourseId(int):List<Grade>
+courseRepository
0..1
+regRepository
0..1
+gradeDAO
0..1
<<Java Interface>>
CourseDAO
springbootthymeleaf.dao
findById(int):Course
deleteById(int):Course
findByProfUsername(String):List<Course>
<<Java Interface>>
RegistrationDAO
springbootthymeleaf.dao
deleteByRegId(RegistrationId):void
deleteAllByRegId_CourseId(int):void
findByRegId(RegistrationId):Registration
findAllByRegId_CourseId(int):List<Registration>
<<Java Interface>>
GradeDAO
springbootthymeleaf.dao
findByRegId(RegistrationId):Grade
deleteByRegId(RegistrationId):void
findAllByRegId_CourseId(int):List<Grade>

Class Name: CourseServiseImpl	
Responsibilities: ▪ This class implements the CourseService interface and describes the operations of the services that are provided by the app.	**Collaborations:** ▪ This class uses the CourseDAO that defines database operations which map domain objects to database table row data. ▪ The CourseMgtAppController uses this class to implement relevant user stories.

Class Name: RegistrationServiceImpl	
Responsibilities: ▪ This class implements the RegistrationService interface and describes the operations of the services that are provided by the app.	**Collaborations:** ▪ This class uses the RegistrationDAO that defines database operations which map domain objects to database table row data. ▪ The CourseMgtAppController uses this class to implement relevant user stories.

Class Name: GradeServiceImpl	
Responsibilities: ▪ This class implements the GradeService interface and describes the operations of the services that are provided by the app.	**Collaborations:** ▪ This class uses the GradeDAO that defines database operations which map domain objects to database table row data. ▪ The CourseMgtAppController uses this class to implement relevant user stories.



<<Java Class>>
CourseMgtAppController
springbootthymeleaf.controller
CourseMgtAppController(CourseService,RegistrationService,GradeService)
getAllCourses(Authentication,Model):String
addCourseForm(Authentication,Model):String
saveCourse(Course,Model):String
showUpdateFormForCourses(int,Model):String
deleteCourse(int):String
getAllRegistrations(int,Model):String
addStudentForm(int,Model):String
saveRegistration(Registration,Model):String
showUpdateFormForReg(int,int,Model):String
editRegistration(Registration,BindingResult,Model):String
deleteRegistration(int,int):String
gradeStudent(int,int,Model):String
saveGrade(Grade,Model):String
listGrades(int,Model):String
calcOverallGrade(int,Model):String
deleteGrade(int,int):String
-courseService
0..1
-registrationService
0..1
-gradeService
0..1
<<Java Interface>>
CourseService
springbootthymeleaf.service
findAll():List<Course>
findByProfUsername(String):List<Course>
findById(int):Course
save(Course):void
deleteById(int):void
<<Java Interface>>
RegistrationService
springbootthymeleaf.service
findAll():List<Registration>
findAllByRegId_CourseId(int):List<Registration>
save(Registration):void
deleteByRegId(RegistrationId):void
deleteAllByRegId_CourseId(int):void
findByRegId(RegistrationId):Registration
<<Java Interface>>
GradeService
springbootthymeleaf.service
findAll():List<Grade>
save(Grade):void
findByRegId(RegistrationId):Grade
deleteByRegId(RegistrationId):void
findAllByRegId_CourseId(int):List<Grade>

Class Name: WebApplicationSecurityConfig	
Responsibilities:	**Collaborations:**
▪ This class is responsible for the security configuration used in the login.	▪ The CourseMgtAppController uses this class to implement relevant user stories.