CoursesManagementApp

Sprint Report

Ομάδα 4387-4399-4508

Κωνσταντίνος Κικίδης – 4387,
Χρήστος Κροκίδας – 4399,
Κωνσταντίνος Τσαμπίρας – 4508

VERSIONS HISTORY

Date	Version	Description	Author
26/3/2022	0.1	Setting up Entity Classes and DB	4387-4399-4508
27/3/2022	0.2	Making the 1st version of our backend	4387-4399-4508
1/4/2022	1.0	Finalizing our prototype, shows the courses of an instructor	4387-4399-4508
2/4/2022	1.1	Adding add/remove/update functionality for courses	4387-4399-4508
5/4/2022	2.0	Setting up our student backend	4387-4399-4508
7/4/2022	2.1	Adding add/remove/update functionality for students, show students button in course view	4387-4399-4508
8/4/2022	2.1.1	Catching some bugs in backend	4387-4399-4508
13/4/2022	2.2	Refining the UI	4387-4399-4508
14/4/2022	2.2.1	Catching some bugs in UI	4387-4399-4508
17/4/2022	3.0	Extending grade functionality (from one grade to project and exam grade)	4387-4399-4508
20/4/2022	3.1	Setting up the final grade calculation and adding UI elements for it	4387-4399-4508
27/4/2022	4.0	Adding calculate stats functionality for exam and project grades (+UI elements)	4387-4399-4508
28/4/2022	4.0.1	Fixing some bugs in median calculation	4387-4399-4508
5/5/2022	5.0	Adding UI login functionality and multiple user support	4387-4399-4508
8/5/2022	6.0	Testing Implementation	4387-4399-4508
9/5/2022	6.0.1	Fixing bugs found	4387-4399-4508
10/5/2022	6.0.2	Fixing more bugs – Final Version	4387-4399-4508

Introduction

1.1 Purpose

Provide the instructors of our institution a way to manage their courses and the students enrolled. The instructor can add grades for their students from there and calculate statistics for their performance in the exam and the project for each course but also generate their final grade on a 70%-Exam 30%-Project weight.

1.2 Document Structure

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

2 Scrum team and Sprint Backlog

2.1 Scrum team

Product Owner	4387, 4399, 4508
Scrum Master	4387, 4399, 4508
Development Team	4387, 4399, 4508

2.2 Sprints

Sprint No	Begin Date	End Date	Number of weeks	User stories
1	**26/3/2022**	**2/4/2022**	**1**	**Create Database and Required Tables, Show Courses for 1 Instructor, Manage Courses (Add/Update/Delete)**
2	**2/4/2022**	**16/4/2022**	**2**	**Show Students for each Course, Manage Students (Add/Update/Delete)**
3	**16/4/2022**	**23/4/2022**	**1**	**Add Grading functionality (for exam and project grade) with the ability to generate the final grade for each student**

4	**23/4/2022**	**30/4/2022**	**1**	**Add Statistics functionality for project and exam grades for a course**
5	**30/4/2022**	**7/5/2022**	**1**	**Make login page for multiple instructors**
6	**7/5/2022**	**14/5/2022**	**1**	**Testing**

3 Use Cases

3.1 LoginPage

Use case ID	1
Actors	Instructor
Pre conditions	Knowing his username and password to login. (Username: john, Password: john / Username: mary, Password: mary123)
Main flow of events	1. The use case starts when the user starts the webpage. 2. The user is requested to provide his username and his password. 3. The system checks if username and password provided are correct. 4. If correct, the system returns the courses of the user who logged in.
Alternative flow	If username/password are incorrect, the system shows a message “Bad credentials”, letting the user know that the credentials he provided are incorrect.
Post conditions	The user can manage his courses safely.

3.2 ShowCourses

Use case ID	2
Actors	Instructor
Pre conditions	User must have successfully logged in.
Main flow of events	1. The use case starts when the user logs in. 2. The system shows a list with his courses.

Alternative flow	User can also go back to the list of courses by pressing the “My Courses” button in the upper right corner.
Post conditions	User can add/change/modify the courses.

3.3 AddCourse

Use case ID	3
Actors	Instructor
Pre conditions	User must have successfully logged in.
Main flow of events	1. The use case starts when the user clicks at the “Add Course” button in the upper-left corner of the page with his courses. 2. The system provides the user with a form to fill with information about the course (Name/Description/Year/Semester). 3. The user fills the form with the necessary information and the clicks the “Save” button. 4. The system returns the list of courses with the new course added.
Alternative flow 1	User provides an alphanumerical value in Year, throws an exception.
Alternative flow 2	User clicks at “Back to Courses” link, going back to the list of courses without adding anything.
Post conditions	A new course is added in the list.

3.4 DeleteCourse

Use case ID	4
Actors	Instructor
Pre conditions	User must have successfully logged in.
Main flow of events	1. The use case starts when the user clicks at the “Delete” button on the course he wants to delete. 2. The system shows an alert to confirm the deletion of the course.

	2.1. If user clicks “OK”, the course is removed from the list. 2.2. If user clicks “Cancel”, nothing happens.
Post conditions	The course is removed from the list.

3.5 UpdateCourse

Use case ID	5
Actors	Instructor
Pre conditions	User must have successfully logged in.
Main flow of events	1. The use case starts when the user clicks at the “Update” button on the course he wants to update. 2. The system provides the user with a form to modify the course he chose (Name/Description/Year/Semester). 3. The user modifies the form and the clicks the “Save” button. 4. The system returns the list of courses with the course selected modified.
Alternative flow 1	User provides an alphanumerical value in Year, throws an exception.
Alternative flow 2	User clicks at “Back to Courses” link, going back to the list of courses without modifying anything.
Post conditions	The course selected is modified.

3.6 ShowStudentsOfACourse

Use case ID	6
Actors	Instructor
Pre conditions	User must have successfully logged in.
Main flow of events	1. The use case starts when the user clicks at the “Show Students” button on the course he wants to see the students enrolled. 2. The system returns a list with the students enrolled in the course selected.
Post conditions	User can add/change/modify the students of the course selected.

3.7 AddStudent

Use case ID	7
Actors	Instructor
Pre conditions	User must have successfully logged in and selected a course.
Main flow of events	1. The use case starts when the user clicks at the “Add Student” button in the upper-left corner of the page with the students of his course. 2. The system provides the user with a form to fill with information about the student (Student ID/Name/Year of Registration). 3. The user fills the form with the necessary information and the clicks the “Save” button. 4. The system returns the list of students of the course with the new student added.
Alternative flow 1	User provides an alphanumerical value in Year of Registration, throws an exception.
Alternative flow 2	User clicks at “Back to Courses” link, going back to the list of courses without adding anything.
Post conditions	A new student is added in the list of students for the course selected

3.8 DeleteStudent

Use case ID	8
Actors	Instructor
Pre conditions	User must have successfully logged in and selected a course.
Main flow of events	1. The use case starts when the user clicks at the “Delete” button on the student he wants to delete. 2. The system shows an alert to confirm the deletion of the course. 2.1. If user clicks “OK”, the course is removed from the list. 2.2. If user clicks “Cancel”, nothing happens.
Post conditions	The student is removed from the list of students for the course selected.

3.9 UpdateStudent

Use case ID	9
Actors	Instructor
Pre conditions	User must have successfully logged in and selected a course.
Main flow of events	1. The use case starts when the user clicks at the “Update Student” button on the student he wants to update his information. 2. The system provides the user with a form to modify the student he chose (Student ID/Name/Year of Registration). 3. The user modifies the form and the clicks the “Save” button. 4. The system returns the list of students for the course selected with the student selected modified.
Alternative flow 1	User provides an alphanumerical value in Year of Registration, throws an exception.
Alternative flow 2	User clicks at “Back to Students List” link, going back to the list of students without modifying anything.
Post conditions	The student selected is modified.

3.10 UpdateGradesOfAStudent

Use case ID	10
Actors	Instructor
Pre conditions	User must have successfully logged in and selected a course.
Main flow of events	1. The use case starts when the user clicks at the “Update Grades” button on the student he wants to update his grades. 2. The system provides the user with a form to update the grades of the students he chose. 3. The user provides the exam and project grades of the student. 4. The system returns the list of students for the course selected with new grades for the student selected.
Alternative flow 1	User provides an alphanumerical value in Project and/or Exam Grade, throws an exception.

Alternative flow 2	User clicks at “Back to Students List” link, going back to the list of students without changing anything.
Post conditions	The student gets his Project and Exam Grade.

3.11 CalculateFinalGradesOfACourse

Use case ID	11
Actors	Instructor
Pre conditions	User must have successfully logged in and selected a course.
Main flow of events	1. The use case starts when the user clicks at the “Calculate Final Grades” button. 2. The system returns a list with the grades for each student, the overall grade they got with a 70%-Exam Grade and 30%-Project Grade weight, and if they passed the course (overall grade >= 5).
Post conditions	The instructor sees the result for each student on his course.

3.12 ShowStatisticsOfACourse

Use case ID	12
Actors	Instructor
Pre conditions	User must have successfully logged in.
Main flow of events	1. The use case starts when the user clicks at the “Show Statistics” button on the course he wants to see the statistics. 2. The system returns a list with statistics on the project grade and exam grade for the students of that course.
Alternative flow	The user can also request to show statistics on the students’ page of the selected course.
Post conditions	The instructor gets some statistical information about the performance of his students.

4 Design

4.1 Architecture

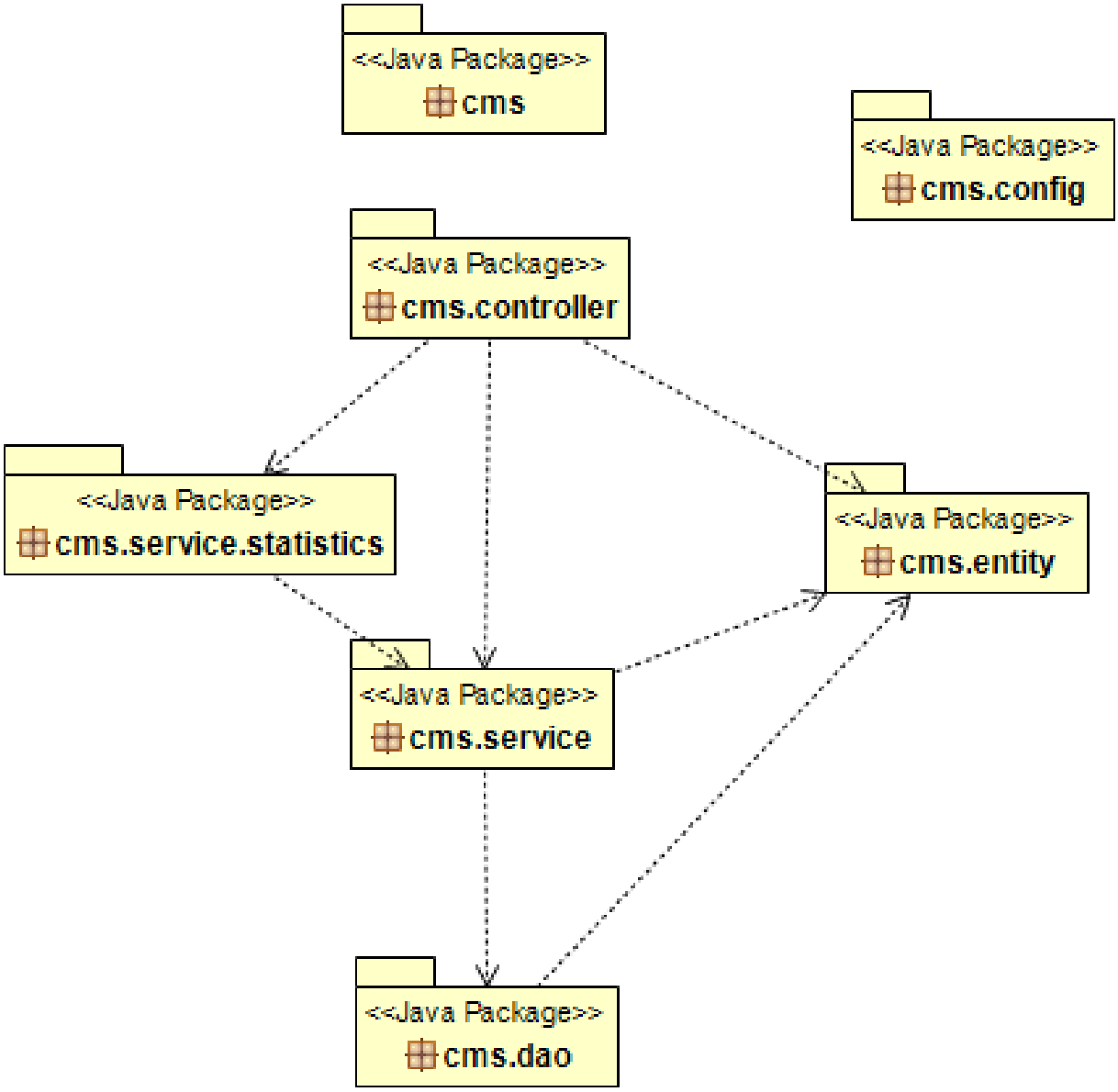


FIGURE 1 PACKAGE DIAGRAM

4.2 Design

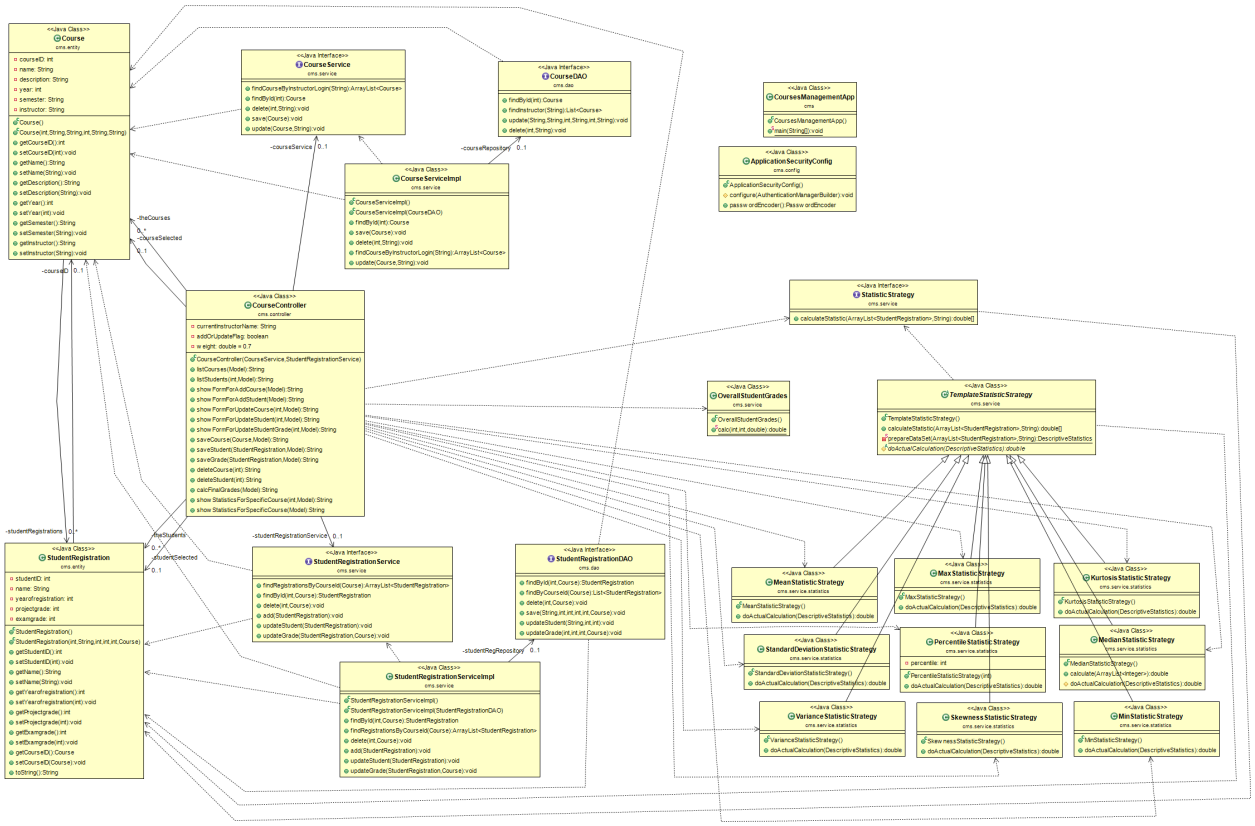


FIGURE 2 CLASS DIAGRAM (GO TO LAST PAGE FOR LARGER IMAGE)

Class Name: CoursesManagementApp	
Responsibilities: ▪ Main Class, starts the SpringBoot Application	**Collaborations:**

Class Name: ApplicationSecurityConfig	
Responsibilities: ▪ Secure login of the instructors	**Collaborations:**

Class Name: CourseController	
Responsibilities: ▪ Controls the UI ▪ Shows the information the instructor requests	**Collaborations:** ▪ Course Entity ▪ Student Registration Entity ▪ Course Service ▪ Student Registration Service

Class Name: CourseService (Interface)	
Responsibilities: ▪ Provides an Interface for the controller	**Collaborations:** ▪ CourseController ▪ CourseServiceImpl

Class Name: CourseServiceImpl	
Responsibilities: ▪ Implements the Interface ▪ Business Logic of the application	**Collaborations:** ▪ CourseService ▪ CourseDAO

Class Name: StudentRegistrationService (Interface)	
Responsibilities: ▪ Provides an Interface for the controller	**Collaborations:** ▪ CourseController ▪ StudentRegistrationServiceImpl

Class Name: StudentRegistrationServiceImpl	
Responsibilities: ▪ Implements the Interface ▪ Business Logic of the application	**Collaborations:** ▪ StudentRegistrationService ▪ StudentRegistrationDAO

Class Name: StatisticStrategy (Interface)	
Responsibilities: ▪ Provides an Interface for the TemplateStatisticStrategy	**Collaborations:** ▪ TemplateStatisticStrategy

Class Name: TemplateStatisticStrategy (Abstract Class)	
Responsibilities: ▪ Implements the Interface ▪ Gets a list of students and prepares the data we need for calculations ▪ Calls the responsible classes	**Collaborations:** ▪ MinStatisticStrategy ▪ MaxStatisticStrategy ▪ MeanStatisticStrategy ▪ MedianStatisticStrategy ▪ SkewnessStatisticStrategy ▪ KurtosisStatisticStrategy ▪ PrecentileStatisticStrategy ▪ StandardDeviationStatisticStrategy ▪ VarianceStatisticStrategy

Class Name: MinStatisticStrategy	
Responsibilities: ▪ Calculates minimum	**Collaborations:** ▪ TemplateStatisticStrategy

Class Name: MaxStatisticStrategy	
Responsibilities: ▪ Calculates maximum	**Collaborations:** ▪ TemplateStatisticStrategy

Class Name: MeanStatisticStrategy	
Responsibilities: ▪ Calculates mean	**Collaborations:** ▪ TemplateStatisticStrategy

Class Name: MedianStatisticStrategy	
Responsibilities: ▪ Calculates median	**Collaborations:** ▪ TemplateStatisticStrategy

Class Name: SkewnessStatisticStrategy	
Responsibilities: ▪ Calculates Skewness	**Collaborations:** ▪ TemplateStatisticStrategy

Class Name: KurtosisStatisticStrategy	
Responsibilities: ▪ Calculates Kurtosis	**Collaborations:** ▪ TemplateStatisticStrategy
Class Name: PercentileStatisticStrategy	
Responsibilities: ▪ Calculates Percentile	**Collaborations:** ▪ TemplateStatisticStrategy

Class Name: StandardDeviationStatisticStrategy	
Responsibilities: ▪ Calculates Standard Deviation	**Collaborations:** ▪ TemplateStatisticStrategy

Class Name: VarianceStatisticStrategy	
Responsibilities: ▪ Calculates Variance	**Collaborations:** ▪ TemplateStatisticStrategy

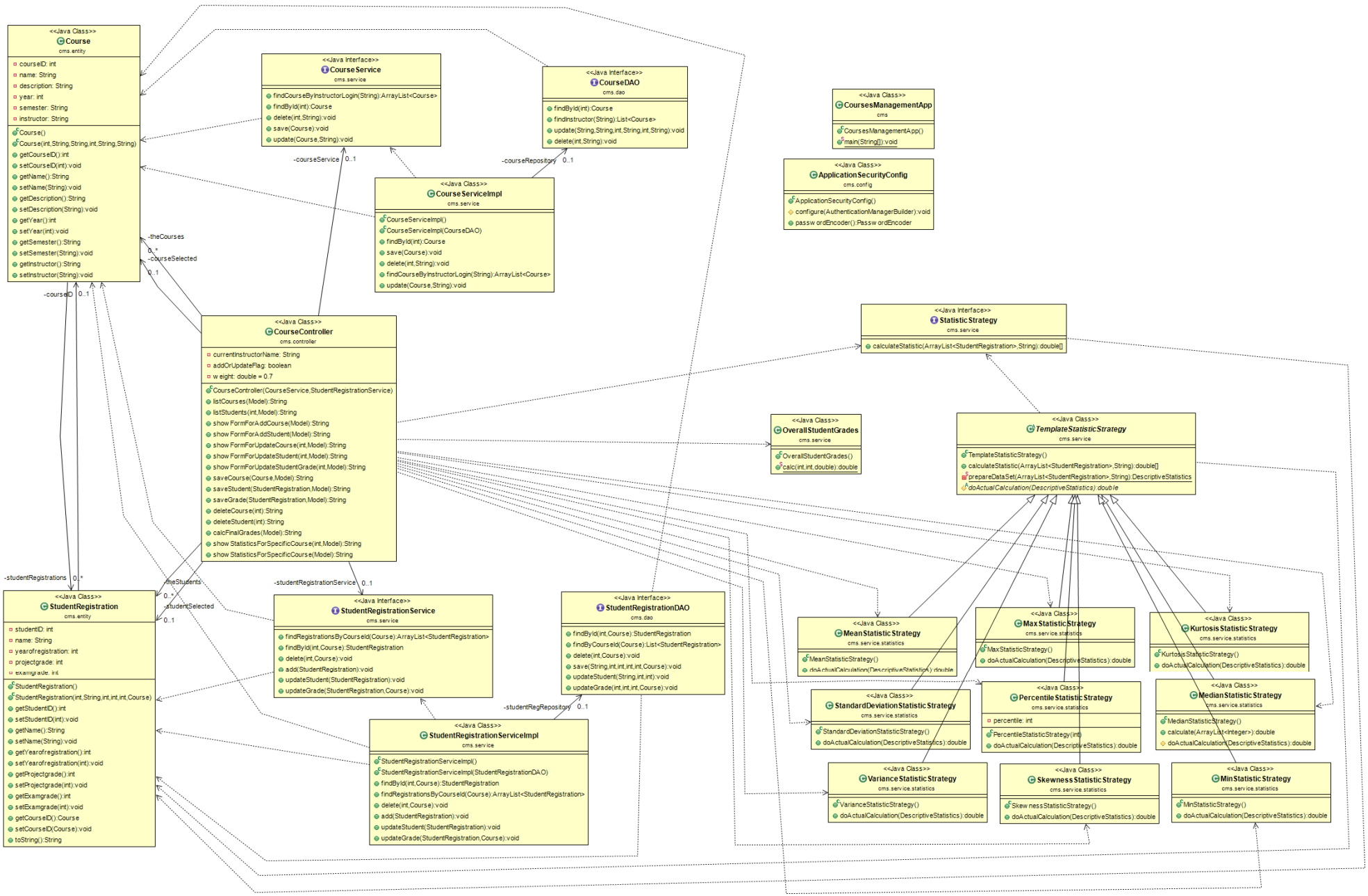
Class Name: OverallStudentGrades	
Responsibilities: ▪ Calculates the overall grades of the students of a course	**Collaborations:** ▪ CourseController

Class Name: CourseDAO (Interface)	
Responsibilities: ▪ Interacts with the DB ▪ Add/Update/Delete Values	**Collaborations:** ▪ CourseServiceImpl

Class Name: StudentRegistrationDAO (Interface)	
Responsibilities: ▪ Interacts with the DB ▪ Add/Update/Delete Values	**Collaborations:** ▪ StudentRegistrationServiceImpl

Class Name: Course	
Responsibilities: ▪ Make Objects from the database table course ▪ Entity Class	**Collaborations:** ▪ CourseController ▪ StudentRegistration

Class Name: StudentRegistration	
Responsibilities: ▪ Make Objects from the database table studentreg ▪ Entity Class	**Collaborations:** ▪ CourseController ▪ Course


<<Java Class>>
Course
cms.entity
courseID: int
name: String
description: String
year: int
semester: String
instructor: String
Course()
Course(int,String,String,int,String,String)
getCourseID():int
setCourseID(int):void
getName():String
setName(String):void
getDescription():String
setDescription(String):void
getYear():int
setYear(int):void
getSemester():String
setSemester(String):void
getInstructor():String
setInstructor(String):void
<<Java Interface>>
CourseService
cms.service
findCourseByInstructorLogin(String):ArrayList<Course>
findById(int):Course
delete(int,String):void
save(Course):void
update(Course,String):void
<<Java Interface>>
CourseDAO
cms.dao
findById(int):Course
findInstructor(String):List<Course>
update(String,String,int,String,int,String):void
delete(int,String):void
<<Java Class>>
CoursesManagementApp
cms
CoursesManagementApp()
main(String[]):void
<<Java Class>>
ApplicationSecurityConfig
cms.config
ApplicationSecurityConfig()
configure(AuthenticationManagerBuilder):void
passwordEncoder():PasswordEncoder
-courseService
0..1
-courseRepository
0..1
<<Java Class>>
CourseServiceImpl
cms.service
CourseServiceImpl()
CourseServiceImpl(CourseDAO)
findById(int):Course
save(Course):void
delete(int,String):void
findCourseByInstructorLogin(String):ArrayList<Course>
update(Course,String):void
-theCourses
0..*
-courseSelected
0..1
-courseID
0..1
<<Java Class>>
CourseController
cms.controller
currentInstructorName: String
addOrUpdateFlag: boolean
weight: double = 0.7
CourseController(CourseService,StudentRegistrationService)
listCourses(Model):String
listStudents(int,Model):String
showFormForAddCourse(Model):String
showFormForAddStudent(Model):String
showFormForUpdateCourse(int,Model):String
showFormForUpdateStudent(int,Model):String
showFormForUpdateStudentGrade(int,Model):String
saveCourse(Course,Model):String
saveStudent(StudentRegistration,Model):String
saveGrade(StudentRegistration,Model):String
deleteCourse(int):String
deleteStudent(int):String
calcFinalGrades(Model):String
showStatisticsForSpecificCourse(int,Model):String
showStatisticsForSpecificCourse(Model):String
<<Java Interface>>
StatisticStrategy
cms.service
calculateStatistic(ArrayList<StudentRegistration>,String):double[]
<<Java Class>>
OverallStudentGrades
cms.service
OverallStudentGrades()
calc(int,int,double):double
<<Java Class>>
TemplateStatisticStrategy
cms.service
TemplateStatisticStrategy()
calculateStatistic(ArrayList<StudentRegistration>,String):double[]
prepareDataSet(ArrayList<StudentRegistration>,String):DescriptiveStatistics
doActualCalculation(DescriptiveStatistics):double
-studentRegistrations
0..*
-theStudents
0..*
-studentSelected
0..1
<<Java Class>>
StudentRegistration
cms.entity
studentID: int
name: String
yearofregistration: int
projectgrade: int
examgrade: int
StudentRegistration()
StudentRegistration(int,String,int,int,int,Course)
getStudentID():int
setStudentID(int):void
getName():String
setName(String):void
getYearofregistration():int
setYearofregistration(int):void
getProjectgrade():int
setProjectgrade(int):void
getExamgrade():int
setExamgrade(int):void
getCourseID():Course
setCourseID(Course):void
toString():String
-studentRegistrationService
0..1
<<Java Interface>>
StudentRegistrationService
cms.service
findRegistrationsByCourseId(Course):ArrayList<StudentRegistration>
findById(int,Course):StudentRegistration
delete(int,Course):void
add(StudentRegistration):void
updateStudent(StudentRegistration):void
updateGrade(StudentRegistration,Course):void
<<Java Interface>>
StudentRegistrationDAO
cms.dao
findById(int,Course):StudentRegistration
findByCourseId(Course):List<StudentRegistration>
delete(int,Course):void
save(String,int,int,int,int,Course):void
updateStudent(String,int,int):void
updateGrade(int,int,int,Course):void
-studentRegRepository
0..1
<<Java Class>>
StudentRegistrationServiceImpl
cms.service
StudentRegistrationServiceImpl()
StudentRegistrationServiceImpl(StudentRegistrationDAO)
findById(int,Course):StudentRegistration
findRegistrationsByCourseId(Course):ArrayList<StudentRegistration>
delete(int,Course):void
add(StudentRegistration):void
updateStudent(StudentRegistration):void
updateGrade(StudentRegistration,Course):void
<<Java Class>>
MeanStatisticStrategy
cms.service.statistics
MeanStatisticStrategy()
doActualCalculation(DescriptiveStatistics):double
<<Java Class>>
MaxStatisticStrategy
cms.service.statistics
MaxStatisticStrategy()
doActualCalculation(DescriptiveStatistics):double
<<Java Class>>
KurtosisStatisticStrategy
cms.service.statistics
KurtosisStatisticStrategy()
doActualCalculation(DescriptiveStatistics):double
<<Java Class>>
StandardDeviationStatisticStrategy
cms.service.statistics
StandardDeviationStatisticStrategy()
doActualCalculation(DescriptiveStatistics):double
<<Java Class>>
PercentileStatisticStrategy
cms.service.statistics
percentile: int
PercentileStatisticStrategy(int)
doActualCalculation(DescriptiveStatistics):double
<<Java Class>>
MedianStatisticStrategy
cms.service.statistics
MedianStatisticStrategy()
calculate(ArrayList<Integer>):double
doActualCalculation(DescriptiveStatistics):double
<<Java Class>>
VarianceStatisticStrategy
cms.service.statistics
VarianceStatisticStrategy()
doActualCalculation(DescriptiveStatistics):double
<<Java Class>>
SkewnessStatisticStrategy
cms.service.statistics
SkewnessStatisticStrategy()
doActualCalculation(DescriptiveStatistics):double
<<Java Class>>
MinStatisticStrategy
cms.service.statistics
MinStatisticStrategy()
doActualCalculation(DescriptiveStatistics):double