CoursesManagementApp

Sprint Report

Team Rocket
Theodorou Ioannis 2444
Tsiaousi Thomai 4510
Zamparas Nikolaos 2969

VERSIONS HISTORY

Date	Version	Description	Author
2/25/2022	<1.0>	Initial design of the application & organizing the requirements in use cases	Team Rocket
3/3/2022	<2.0>	Organization & started developing core classes	Team Rocket
3/26/2022	<3.0>	Development & optimization of the app	Team Rocket
5/2/2022	<4.0>	Testing	Team Rocket
5/12/2022	<5.0>	Final Report	Team Rocket

1 Introduction

This document provides information concerning the 1,2,3,4 sprints of the project.

1.1 Purpose

The objective of this project is the development of a Web application that allows an instructor to manage the courses he teaches along with the students that are registered in these courses and their grading.

1.2 Document Structure

The rest of this document is structured as follows. Section 2 describes out Scrum team and specifies the 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	Nikos Zamparas, John Theodorou, Thomai Tsiaousi
Scrum Master	Nikos Zamparas, John Theodorou, Thomai Tsiaousi
Development Team	Nikos Zamparas, John Theodorou, Thomai Tsiaousi

2.2 Sprints

Sprint No	Begin Date	End Date	Number of weeks	User stories
1	**3/3/2022**	**3/24/2022**	**3**	**US2, US3, US4, US5**
2	**3/25/2022**	**4/8/2022**	**2**	**US6, US7, US8, US9**
3	**4/9/2022**	**4/16/2022**	**1**	**US1**
4	**4/17/2022**	**5/1/2022**	**2**	**US10, US11, US12**

3 Use Cases

3.1 LogIn

Use case ID	UC1
Actors	Instructor
Pre conditions	The instructor has a username and password.
Main flow of events	1. The use case starts when the instructor wants to login to the program. 2. The system determines if the username and the password are correct.
Alternative flow	Instructor username and/ or password are incorrect.
Post conditions	The instructor can see the list of courses he teaches (Course Directory).

3.2 Browse Courses

Use case ID	UC2
Actors	Instructor
Pre conditions	The instructor has logged in the system
Main flow of events	1. The use case starts when the instructor chooses to browse the courses he teaches. 2. The system interacts with the instructor's choices (scroll up and down, click on courses, go back to the courses list again).
Post conditions	The instructor can browse through the list of courses he teaches (Course Directory).

3.3 Add Course

Use case ID	UC3
Actors	Instructor
Preconditions	The system must use a database schema to store all the information concerning a course.
Main flow of events	1. The use case starts when the instructor is about to add a new course in his teaching course list. 2. The instructor gives the attributes of the course like course id, name, syllabus, year, semester, etc. 3. The system stores the course in the teaching course list (database).
Post conditions	The course is in the list of instructor's courses.

3.4 Delete Course

Use case ID	UC4
Actors	Instructor
Preconditions	The system must use a database schema to store all the information concerning a course.
Main flow of events	1. The use case starts when the instructor is about to delete a course from his teaching course list. 2. The instructor interacts with corresponding delete icon and deletes the course. 3. The system deletes the course in the teaching course list (database).
Post conditions	The course is no longer in the database of the instructor's courses.

3.5 Update Description Of Course

Use case ID	UC5
Actors	Instructor
Preconditions	Course list shouldn't be empty.
Main flow of events	1. The use case starts when the instructor chooses to update the course's description and clicks the update icon. 2. The instructor makes the appropriate changes to the syllabus's course field. 3. The system stores all the changes to the course.
Post conditions	The description of the course has changed successfully.

3.6 Browse List Of Students

Use case ID	UC6
Actors	Instructor
Preconditions	The system must use a database schema to store all the information concerning a student registration for each course.
Main flow of events	1. The use case starts when the instructor chooses to browse the list of the students to a particular course. 2. The system interacts with instructor's choice to click on the show registrations button.
Post conditions	The instructor can see the list of the students registered in the chosen course.

3.7 Add Student

Use case ID	UC7
Actors	Instructor
Preconditions	The system must use a database schema to store all the information concerning a student registration for each course.
Main flow of events	1. The use case starts when the instructor chooses to add a student in the course. 2. The system interacts with instructor's choice to click on the Add Student button. 3. The instructor gives the attribute of the student like student id, name, year of registration, semester, etc. 4. The system stores the student in the teaching course/student list (database).
Post conditions	The student is now stored in the teaching course/student list(database).

3.8 Delete Student

Use case ID	UC8
Actors	Instructor
Preconditions	The system must use a database schema to store all the information concerning a student registration for each course.
Main flow of events	1. The use case starts when the instructor chooses to remove a student from the course. 2. The system interacts with instructor's choice to click on the Delete button. 3. The system deletes the student from the corresponding course/student list (database).
Post conditions	The student is now deleted from the teaching course/student list(database).

3.9 Update Student Information

Use case ID	UC9
Actors	Instructor
Preconditions	The system must use a database schema to store all the information concerning a student registration for each course.
Main flow of events	1. The use case starts when the instructor chooses to update a student's info in the course. 2. The system interacts with instructor's choice to click on the Update button. 3. The instructor changes either id, name, year of semester, etc. 4. The system on click -Save Student- Updates the students info.
Post conditions	The student is updated in the teaching course/student list (database).

3.10 Register Grades

Use case ID	UC10
Actors	Instructor
Preconditions	The system must use a database schema to store all the information concerning a student registration for each course.
Main flow of events	1. The use case starts when the instructor chooses to register the project and exam grades of a student.
Post conditions	The student's grades of a course are now registered.

3.11 Calculate Weighted Average

Use case ID	UC11
Actors	Instructor
Preconditions	Each student must have a registered grade.
Main flow of events	1. The use case starts when, given a weighted average, the system calculates the overall grades of the students enrolled in a particular course.
Alternative flow	The students have not be graded yet.
Post conditions	The overall grades of the students are now calculated.

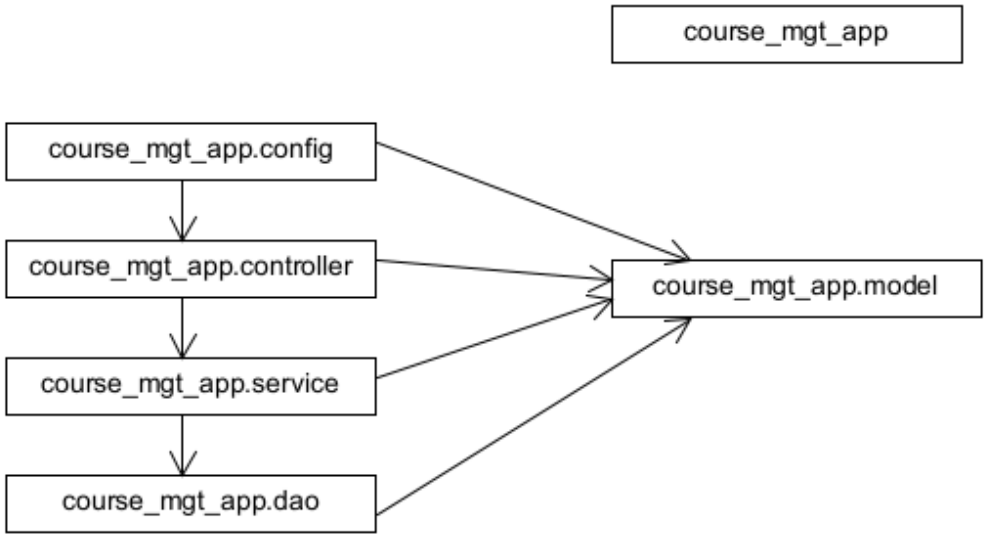
3.12 Calculate Descriptive Statistics

Use case ID	UC12
Actors	Instructor
Preconditions	Each student must have a registered grade.
Main flow of events	1. The use case starts when the instructor clicks on the -Course Statistics- button. 2. The system calculates the descriptive statistics about the students in a particular course.
Alternative flow	There aren't any grades registered to the students.
Post conditions	The statistics about the students in a course are now calculated.

4 Design

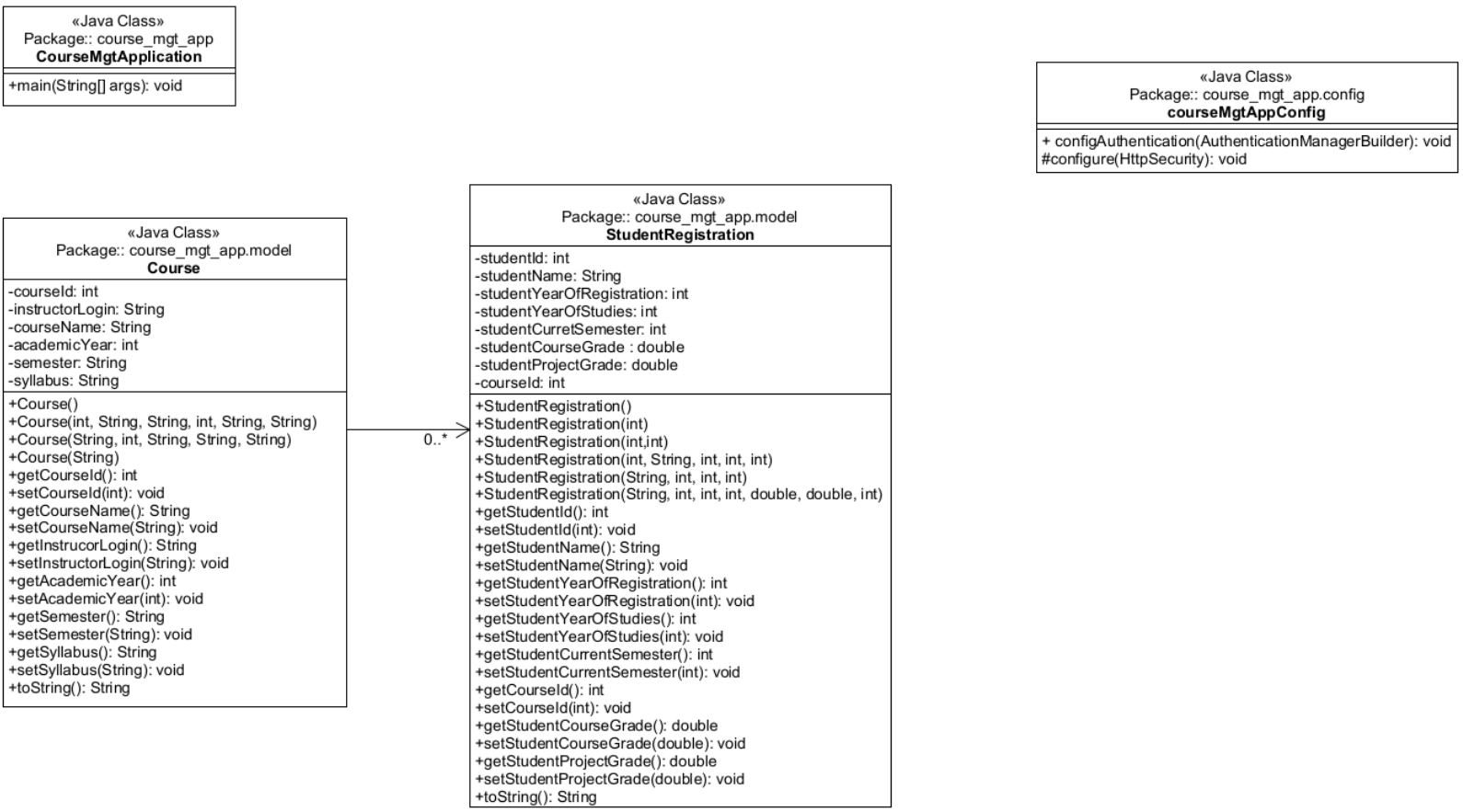
4.1 Architecture

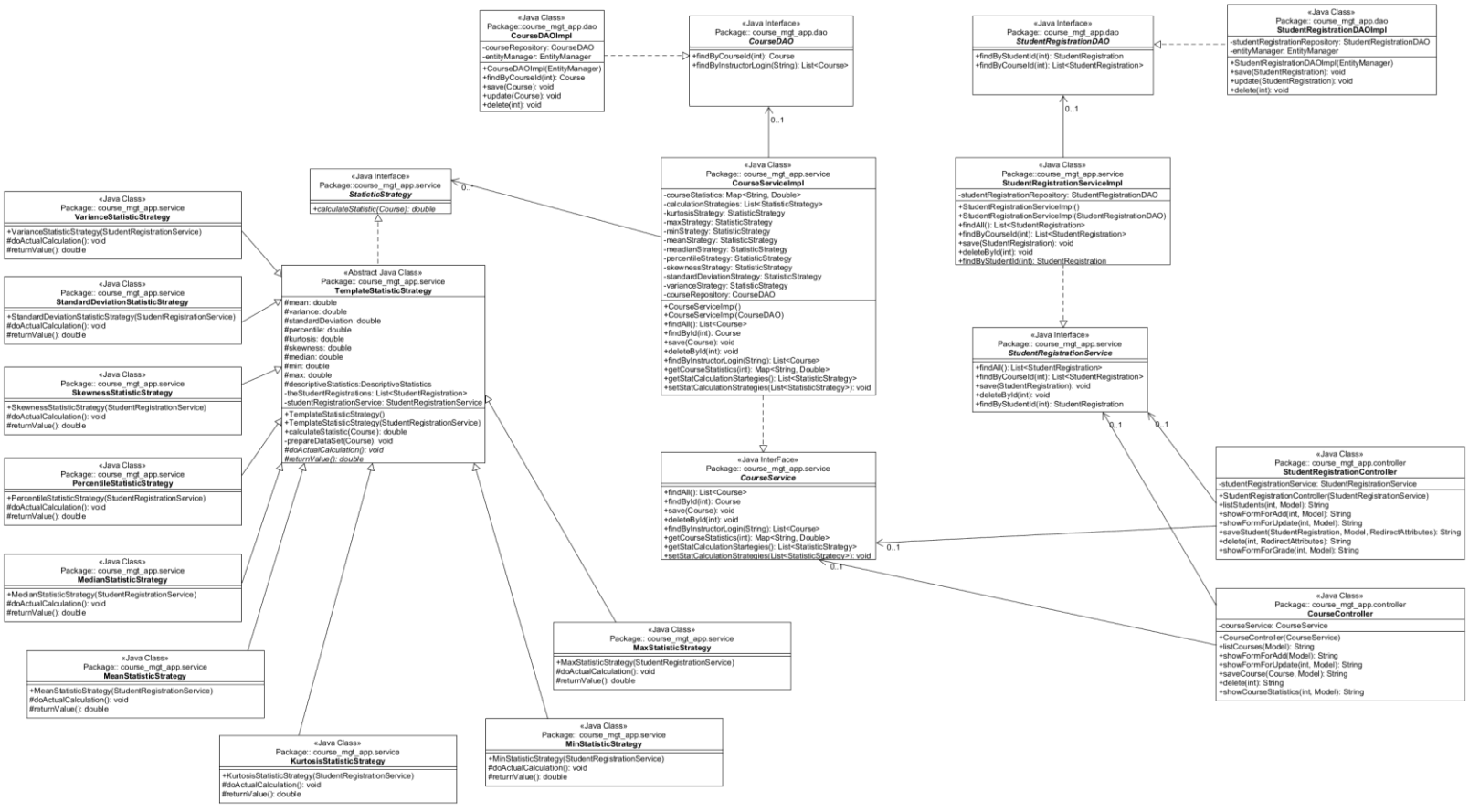
UML package diagram



4.2 Design

UML class diagrams




«Java Class»
Package:: course_mgt_app.dao
CourseDAOImpl
-courseRepository: CourseDAO
-entityManager: EntityManager
+CourseDAOImpl(EntityManager)
+findByCourseId(int): Course
+save(Course): void
+update(Course): void
+delete(int): void
«Java Interface»
Package:: course_mgt_app.dao
CourseDAO
+findByCourseId(int): Course
+findByInstructorLogin(String): List<Course>
«Java Interface»
Package:: course_mgt_app.dao
StudentRegistrationDAO
+findByStudentId(int): StudentRegistration
+findByCourseId(int): List<StudentRegistration>
«Java Class»
Package:: course_mgt_app.dao
StudentRegistrationDAOImpl
-studentRegistrationRepository: StudentRegistrationDAO
-entityManager: EntityManager
+StudentRegistrationDAOImpl(EntityManager)
+save(StudentRegistration): void
+update(StudentRegistration): void
+delete(int): void
0..1
0..1
«Java Interface»
Package:: course_mgt_app.service
StaticticStrategy
+calculateStatistic(Course): double
0..*
«Java Class»
Package:: course_mgt_app.service
CourseServiceImpl
-courseStatistics: Map<String, Double>
-calculationStrategies: List<StatisticStrategy>
-kurtosisStrategy: StatisticStrategy
-maxStrategy: StatisticStrategy
-minStrategy: StatisticStrategy
-meanStrategy: StatisticStrategy
-medianStrategy: StatisticStrategy
-percentileStrategy: StatisticStrategy
-skewnessStrategy: StatisticStrategy
-standardDeviationStrategy: StatisticStrategy
-varianceStrategy: StatisticStrategy
-courseRepository: CourseDAO
+CourseServiceImpl()
+CourseServiceImpl(CourseDAO)
+findAll(): List<Course>
+findById(int): Course
+save(Course): void
+deleteById(int): void
+findByInstructorLogin(String): List<Course>
+getCourseStatistics(int): Map<String, Double>
+getStatCalculationStartegies(): List<StatisticStrategy>
+setStatCalculationStrategies(List<StatisticStrategy>): void
«Java Class»
Package::course_mgt_app.service
StudentRegistrationServiceImpl
-studentRegistrationRepository: StudentRegistrationDAO
+StudentRegistrationServiceImpl()
+StudentRegistrationServiceImpl(StudentRegistrationDAO)
+findAll(): List<StudentRegistration>
+findByCourseId(int): List<StudentRegistration>
+save(StudentRegistration): void
+deleteById(int): void
+findByStudentId(int): StudentRegistration
«Java Interface»
Package:: course_mgt_app.service
StudentRegistrationService
+findAll(): List<StudentRegistration>
+findByCourseId(int): List<StudentRegistration>
+save(StudentRegistration): void
+deleteById(int): void
+findByStudentId(int): StudentRegistration
«Java Class»
Package:: course_mgt_app.service
VarianceStatisticStrategy
+VarianceStatisticStrategy(StudentRegistrationService)
#doActualCalculation(): void
#returnValue(): double
«Java Class»
Package:: course_mgt_app.service
StandardDeviationStatisticStrategy
+StandardDeviationStatisticStrategy(StudentRegistrationService)
#doActualCalculation(): void
#returnValue(): double
«Java Class»
Package:: course_mgt_app.service
SkewnessStatisticStrategy
+SkewnessStatisticStrategy(StudentRegistrationService)
#doActualCalculation(): void
#returnValue(): double
«Java Class»
Package:: course_mgt_app.service
PercentileStatisticStrategy
+PercentileStatisticStrategy(StudentRegistrationService)
#doActualCalculation(): void
#returnValue(): double
«Java Class»
Package:: course_mgt_app.service
MedianStatisticStrategy
+MedianStatisticStrategy(StudentRegistrationService)
#doActualCalculation(): void
#returnValue(): double
«Java Class»
Package:: course_mgt_app.service
MeanStatisticStrategy
+MeanStatisticStrategy(StudentRegistrationService)
#doActualCalculation(): void
#returnValue(): double
«Abstract Java Class»
Package:: course_mgt_app.service
TemplateStatisticStrategy
#mean: double
#variance: double
#standardDeviation: double
#percentile: double
#kurtosis: double
#skewness: double
#median: double
#min: double
#max: double
#descriptiveStatistics:DescriptiveStatistics
-theStudentRegistrations: List<StudentRegistration>
-studentRegistrationService: StudentRegistrationService
+TemplateStatisticStrategy()
+TemplateStatisticStrategy(StudentRegistrationService)
+calculateStatistic(Course): double
-prepareDataSet(Course): void
#doActualCalculation(): void
#returnValue(): double
«Java InterFace»
Package:: course_mgt_app.service
CourseService
+findAll(): List<Course>
+findById(int): Course
+save(Course): void
+deleteById(int): void
+findByInstructorLogin(String): List<Course>
+getCourseStatistics(int): Map<String, Double>
+getStatCalculationStartegies(): List<StatisticStrategy>
+setStatCalculationStrategies(List<StatisticStrategy>): void
0..1
0..1
0..1
0..1
«Java Class»
Package:: course_mgt_app.controller
StudentRegistrationController
-studentRegistrationService: StudentRegistrationService
+StudentRegistrationController(StudentRegistrationService)
+listStudents(int, Model): String
+showFormForAdd(int, Model): String
+showFormForUpdate(int, Model): String
+saveStudent(StudentRegistration, Model, RedirectAttributes): String
+delete(int, RedirectAttributes): String
+showFormForGrade(int, Model): String
«Java Class»
Package:: course_mgt_app.controller
CourseController
-courseService: CourseService
+CourseController(CourseService)
+listCourses(Model): String
+showFormForAdd(Model): String
+showFormForUpdate(int, Model): String
+saveCourse(Course, Model): String
+delete(int): String
+showCourseStatistics(int, Model): String
«Java Class»
Package:: course_mgt_app.service
MaxStatisticStrategy
+MaxStatisticStrategy(StudentRegistrationService)
#doActualCalculation(): void
#returnValue(): double
«Java Class»
Package:: course_mgt_app.service
KurtosisStatisticStrategy
+KurtosisStatisticStrategy(StudentRegistrationService)
#doActualCalculation(): void
#returnValue(): double
«Java Class»
Package:: course_mgt_app.service
MinStatisticStrategy
+MinStatisticStrategy(StudentRegistrationService)
#doActualCalculation(): void
#returnValue(): double

1. **Model Layer Classes:**

Class Name: Course	
Responsibilities: ▪ Knows its course ID ▪ Knows its instructor Login ▪ Knows its course Name ▪ Knows its semester ▪ Knows its academic Year ▪ Knows its syllabus	**Collaborations:**

Class Name: StudentRegistration	
Responsibilities: ▪ Knows its Course ID ▪ Knows its Student ID ▪ Knows its Student Name ▪ Knows its Year of Registration ▪ Knows its Current Semester ▪ Knows its Course Grade ▪ Knows its Project Grade	**Collaborations:** ▪ Course

2. DAO Layer Classes:

Class Name: CourseDAO	
Responsibilities: ▪ Finds a Course by the Course ID ▪ Finds a List of the Instructor's Courses	**Collaborations:** ▪ Course, JPARepository ▪ Course, JPARepository

Class Name: StudentRegistrationDAO	
Responsibilities: ▪ Finds a Student Registration by the Student's ID ▪ Finds the List of Student Registrations of a specified Course (by its Course ID)	**Collaborations:** ▪ StudentRegistration, JPARepository, studentId ▪ StudentRegistration, JPARepository, courseId

3. Service Layer Classes:

Class Name: CourseServiceImpl	
Responsibilities: ▪ Finds a Course by its ID ▪ Finds a List of the Instructor's Courses ▪ Saves a Course to the Database ▪ Deletes a Course from the Database by its ID ▪ Calculates the Statistics of a Course ▪ Allows to get the Statistic Strategies of a Course ▪ Allows to set the Statistic Strategies of a Course	**Collaborations:** ▪ CourseDAO, courseId ▪ CourseDAO, instructorLogin ▪ CourseDAO, Course ▪ CourseDAO, courseId ▪ CourseId, StatisticStrategy ▪ StatisticStrategy ▪ StatisticStrategy

Class Name: StudentRegistrationServiceImpl	
Responsibilities: ▪ Finds a Student Registration by its ID ▪ Finds the List of Student Registrations of a Course by the Course's ID ▪ Saves a Student Registration to the Database ▪ Deletes a Student Registration from the Database	**Collaborations:** ▪ StudentRegistrationDAO, StudentRegistration, studentId ▪ StudentRegistrationDAO, StudentRegistration, courseId ▪ StudentRegistrationDAO, StudentRegistration ▪ StudentRegistrationDAO, studentId

Class Name: TemplateStatisticStrategy	
Responsibilities: ▪ Calculates the Statistics of a Course's Grades	**Collaborations:** ▪ DescriptiveStatistics, StudentRegistrationService, mean, variance, standardDeviation, percentile, kurtosis, skewness, median, min, max

Class Name: KurtosisStatisticStrategy	
Responsibilities: ▪ Calculates the Kurtosis of a Course's Grades	**Collaborations:** ▪ DescriptiveStatistics, StudentRegistrationService, kurtosis

Class Name: MaxStatisticStrategy	
Responsibilities: ▪ Calculates the Maximum Value of a Course's Grades	**Collaborations:** ▪ DescriptiveStatistics, StudentRegistrationService, max

Class Name: MinStatisticStrategy	
Responsibilities: ▪ Calculates the Minimum Value of a Course's Grades	**Collaborations:** ▪ DescriptiveStatistics, StudentRegistrationService, min

Class Name: MeanStatisticStrategy	
Responsibilities: ▪ Calculates the Mean Value of a Course's Grades	**Collaborations:** ▪ DescriptiveStatistics, StudentRegistrationService, mean

Class Name: MedianStatisticStrategy	
Responsibilities: ▪ Calculates the Median of a Course's Grades	**Collaborations:** ▪ DescriptiveStatistics, StudentRegistrationService, median

Class Name: PercentileStatisticStrategy	
Responsibilities: ▪ Calculates a Percentile of a Course's Grades	**Collaborations:** ▪ DescriptiveStatistics, StudentRegistrationService, percentile

Class Name: SkewnessStatisticStrategy	
Responsibilities: ▪ Calculates the Skewness of a Course's Grades	**Collaborations:** ▪ DescriptiveStatistics, StudentRegistrationService, skewness

Class Name: StandardDeviationStatisticStrategy	
Responsibilities: ▪ Calculates the Standard Deviation of a Course's Grades	**Collaborations:** ▪ DescriptiveStatistics, StudentRegistrationService, standardDeviation

4. Controller Layer Classes:

Class Name: CourseController	
Responsibilities: ▪ Shows the List of an Instructor's Courses ▪ Shows a Form for Adding a Course ▪ Shows a Form for Updating a Course ▪ Saves the contents of a Course Form ▪ Deletes a Course from the List	**Collaborations:** ▪ Model, instructorLogin, Course, Authentication ▪ Authentication, Course ▪ courseId, Model, Course ▪ CourseService, courseId, Course, Model ▪ CourseId, CourseService

Class Name: StudentRegistrationController	
Responsibilities: ▪ Shows the List of a Course's Student Registrations ▪ Shows a Form for Adding a Student Registration to a Course ▪ Shows a Form for Updating a Student Registration ▪ Saves the contents of a Student Registration Form ▪ Deletes a Student Registration from the List	**Collaborations:** ▪ courseId, Model, StudentRegistrationService, StudentRegistration, Course ▪ courseId, Model, StudentRegistration ▪ studentId, Model, StudentRegistration ▪ StudentRegistrationService, Model, ▪ StudentId, StudentRegistration, StudentRegistrationService

5. **Main Class:**

Class Name: CourseMgtApp	
Responsibilities: ▪ Runs the Application	**Collaborations:** ▪ SpringApplication, SpringBoot