CoursesManagementApp

Sprint Report

SOTIRIOS PANAGIOTOU 4456, cs04456@uoi.gr
MYRITZIS EFSTRATIOS 4444, cs04444@uoi.gr
DIMITRIOS GIANNAKOPOULOS 4336, cs04336@uoi.gr

*Για να τρέξουν τα statistics πρέπει να γίνει import του commons-math3.jar library

Github: https://github.com/sotblad/SE-Project

Demo video: https://github.com/sotblad/SE-Project/blob/main/zarr.mp4

VERSIONS HISTORY

Date	Version	Description	Author
15/5/22	1.0	User stories implemented. Unit tests all pass without any errors. Everything engineered with maintainability and usability in mind.	SOTIRIOS PANAGIOTOU MYRITZIS EFSTATIOS DIMITRIS GIANNAKOPOULOS

1 Introduction

This document provides information concerning the **3** sprints of the project.

The objective of this project is to develop a Web application that allows an instructor to manage the grading of the courses that he teaches.

1.1 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	Zarras
Scrum Master	-
Development Team	SOTIRIOS PANAGIOTOU MYRITZIS EFSTATIOS DIMITRIS GIANNAKOPOULOS

2.2 Sprints

Sprint No	Begin Date	End Date	Number of weeks	User stories
S1	**3/5/22**	**4/5/22**	**0**	**US1,US1.1,US2,US3,US4,US5**
S2	**4/5/22**	**5/5/22**	**0**	**US6,US7,US8,US9,US10**
S3	**5/5/22**	**6/5/22**	**0**	**US11,US12**

3 Use Cases

3.1 <Use Case 1>

Use case ID	1
Actors	Instructor
Pre conditions	For the correct execution of US1, a database has to be created and inside it there must be stored the user names and passwords of the users.
Main flow of events	1. The use case starts when the user loads the /login page. 2. There is a form where the user is able to fill in his info(username, password).
Post conditions	After the correct execution of this use case, the user has access to the list of courses being taught by him.

3.1.1 <Use Case 1.1>

Use case ID	1.1
Actors	Instructor
Pre conditions	For the correct execution of US1.1, US1 has to be implemented. And in the /login page there is a button for the user to be redirected in the /register page.
Main flow of events	1. The use case starts when the user loads the /register page. 2. There is a form where the user must be able to fill in his info. 3. The info of the new user is added to the user database.
Post conditions	After the correct execution of this use case, the user is redirected back to the /login page .

3.2 <Use Case 2>

Use case ID	2
Actors	Instructor
Pre conditions	For the correct execution of US2, US1 has to be implemented.
Main flow of events	1. The use case starts after inserting his credentials in the login form and being redirected in the /myCourses page. 2. On each course there are buttons to: 2.1 View the course, 2.2 Edit the course info, 2.3 Delete the course.
Post conditions	After the correct execution of this use case, the user is able to manage the courses being taught by him.

3.3 <Use Case 3>

Use case ID	3
Actors	Instructor
Pre conditions	For the correct execution of US3, US1 and US2 have to be implemented.
Main flow of events	1. The use case starts after clicking the blue “Add Course” button in the /myCourses page. 2. Then the user is redirected to the /addCourse page where he is asked to fill in the following information about the course: 2.1 Course name, 2.2 Syllabus, 2.3 Year, 2.4 Semester.
Post conditions	After the correct execution of this use case, the new course is added to the instructor’s courses.

3.4 <Use Case 4>

Use case ID	4
Actors	Instructor
Pre conditions	For the correct execution of US4, US1, US2 and US3 have to be implemented.
Main flow of events	1. The use case starts after clicking the red “Trash Can” button in the /myCourses page.
Post conditions	After the correct execution of this use case, the selected course is removed from the courses being taught by the instructor.

3.5 <Use Case 5>

Use case ID	5
Actors	Instructor
Pre conditions	For the correct execution of US5, US1, US2 and US3 have to be implemented.
Main flow of events	1. The use case starts after clicking the green button in the /myCourses page. 2. Then the user is redirected in the /editCourse page where he is asked to edit the following information about the course: 2.1 Course name, 2.2 Syllabus, 2.3 Year, 2.4 Semester.
Post conditions	After the correct execution of this use case, the selected course’s information is updated.

3.6 <Use Case 6>

Use case ID	6
Actors	Instructor
Pre conditions	For the correct execution of US6, US1, US2 and US3 have to be implemented.
Main flow of events	1. The use case starts after clicking the blue “Arrow” button in the /myCourses page.
Post conditions	After the correct execution of this use case, the instructor is redirected to the /viewCourse page of the selected course where there is a list of students enrolled in the specific course.

3.7 <Use Case 7>

Use case ID	7
Actors	Instructor
Pre conditions	For the correct execution of US7, US1, US2, US3 and US6 have to be implemented.
Main flow of events	1. The use case starts after clicking the blue “Add student” button in the /viewCourse page. 2. Then the user is redirected in the /addStudent page where he is asked to fill in the information of the new student: 2.1 Student ID, 2.2 Student Name, 2.3 Year of registration, 2.4 Semester.
Post conditions	After the correct execution of this use case, the instructor is redirected back to the selected course where he should be able to see the new student.

3.8 <Use Case 8>

Use case ID	8
Actors	Instructor
Pre conditions	For the correct execution of US8, US1, US2, US3, US6 and US7 have to be implemented.
Main flow of events	1. The use case starts after clicking the red “Delete” button in the /viewCourse page next to a student.
Post conditions	After the correct execution of this use case, the instructor is redirected back to the selected course where the deleted student should be missing.

3.9 <Use Case 9>

Use case ID	9
Actors	Instructor
Pre conditions	For the correct execution of US9, US1, US2, US3, US6 and US7 have to be implemented.
Main flow of events	1. The use case starts after clicking the blue “Edit Student” button in the /viewCourse page. 2. Then the instructor is redirected in the /editStudent page where he is asked to edit the following information about the student: 2.1 Student ID, 2.2 Student Name, 2.3 Year of registration, 3. Finally the user has the option to either Soft update or Hard update. 3.1. When Soft updating, the student’s new information is only updated in the specific course page. 3.2. When Hard updating, the student’s new information is updated in every course the student participates.
Post conditions	After the correct execution of this use case, the instructor is redirected back to the selected course where the student’s information is updated with the new values.

3.10 <Use Case 10>

Use case ID	10
Actors	Instructor
Pre conditions	For the correct execution of US10, US1, US2, US3, US6 and US7 have to be implemented.
Main flow of events	1. The use case starts after clicking the green “Edit Grades” button in the /viewCourse page. 2. Then the user is redirected to the /editGrades page where he is asked to edit the following information about the student: 2.1 Exam grade, 2.2 Project grade
Post conditions	After the correct execution of this use case, the instructor is redirected back to the selected course and the student’s grades are updated with the new values.

3.11 <Use Case 11>

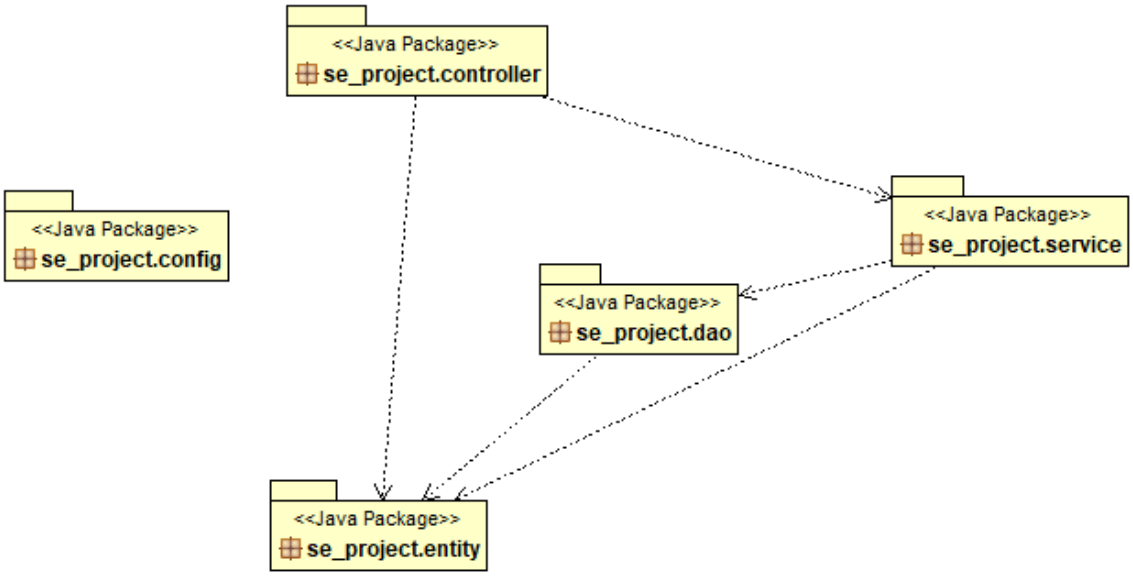
Use case ID	11
Actors	Instructor
Pre conditions	For the correct execution of US11, US1, US2, US3, US6, US7 and US10 have to be implemented. The user must have clicked the green “Edit Weights” button and have entered the exam-project weights. Also the instructor should have pressed the green “Edit Grades” button to enter the corresponding grades.
Main flow of events	1. The use case starts after clicking the blue “Calculate Grades” button in the /viewCourse page.
Post conditions	After the correct execution of this use case, the user is on the /viewCourse page where the student’s grade is updated with the generated values.

3.12 <Use Case 12>

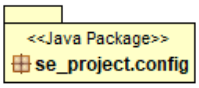
Use case ID	12
Actors	Instructor
Pre conditions	For the correct execution of US12, US1, US2, US3, US6, US7, US10 and US11 have to be implemented.
Main flow of events	1. The use case starts after clicking the red “View Stats” button in the /viewCourse page. 2. The instructor is redirected to the /viewStats page where he can see the generated descriptive statistics about the grades of the students enrolled in the course. The statistics include: 2.1 Min.Grade 2.2 Max. Grade 2.3 Mean Grade 2.4 Std. Deviation 2.5 Variance 2.6 Skewness 2.7 Kurtosis 2.8 Median 2.9 Percentiles
Post conditions	After the execution of this use case, the statistics get calculated and displayed on the /viewStats page of the selected course.

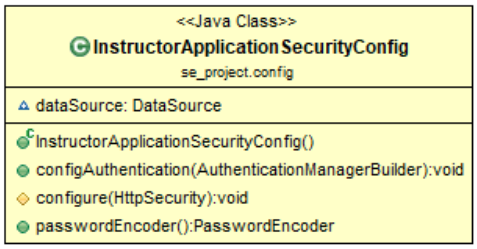
4 Design

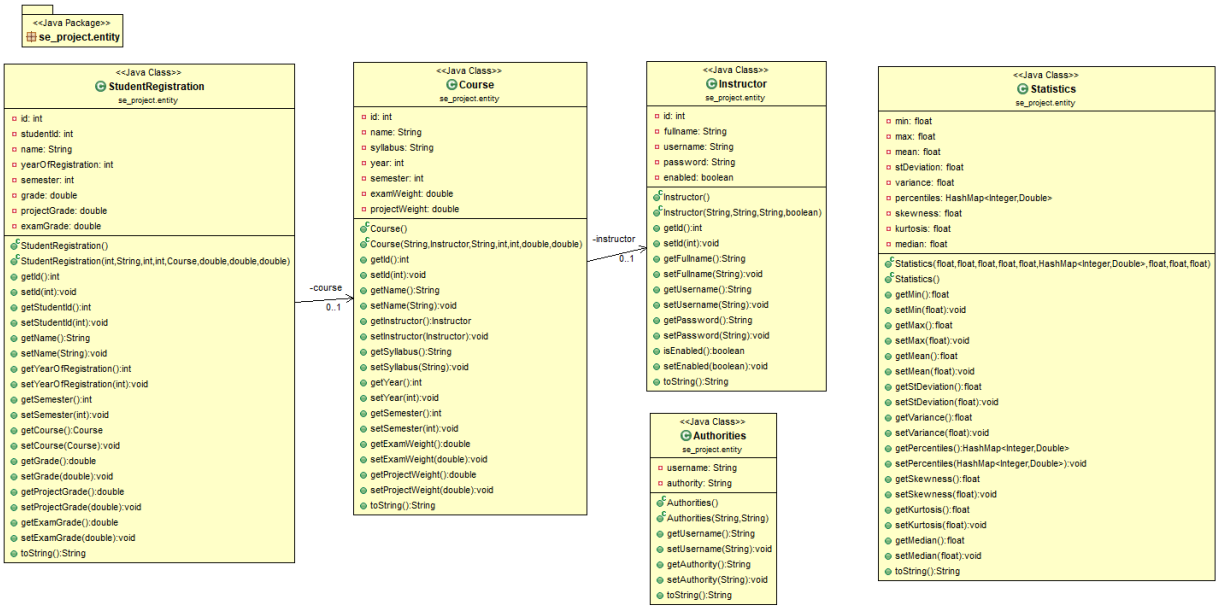
4.1 Architecture

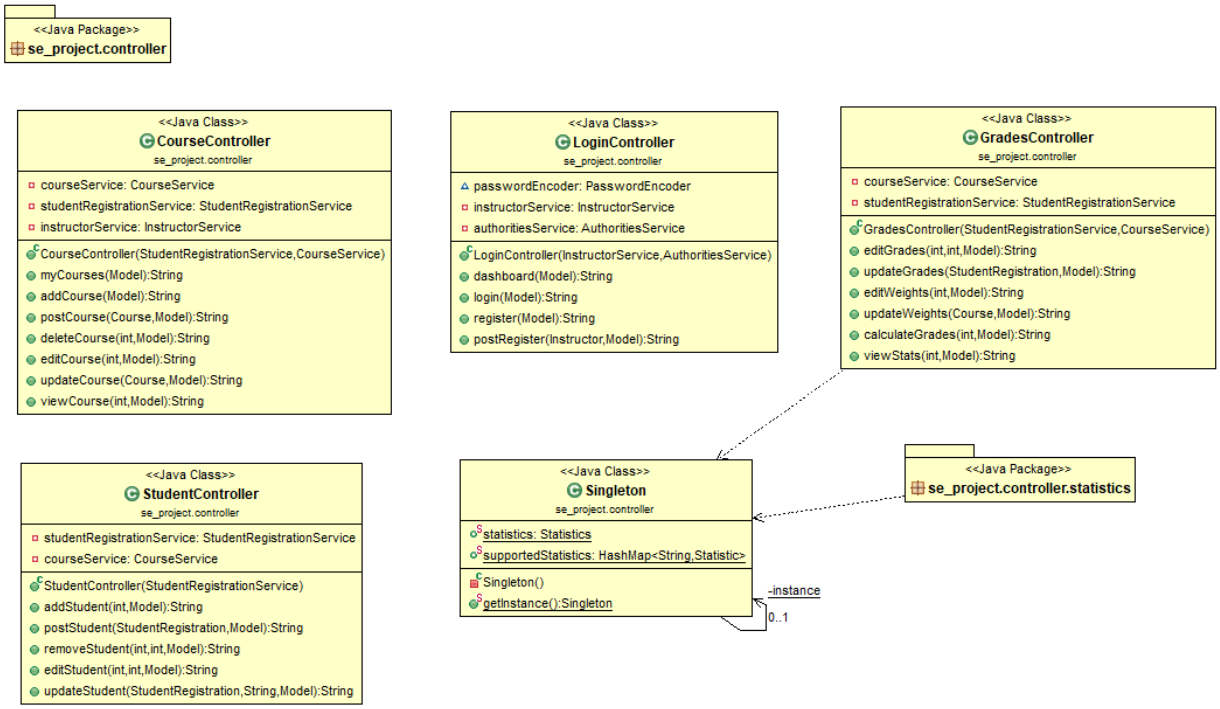


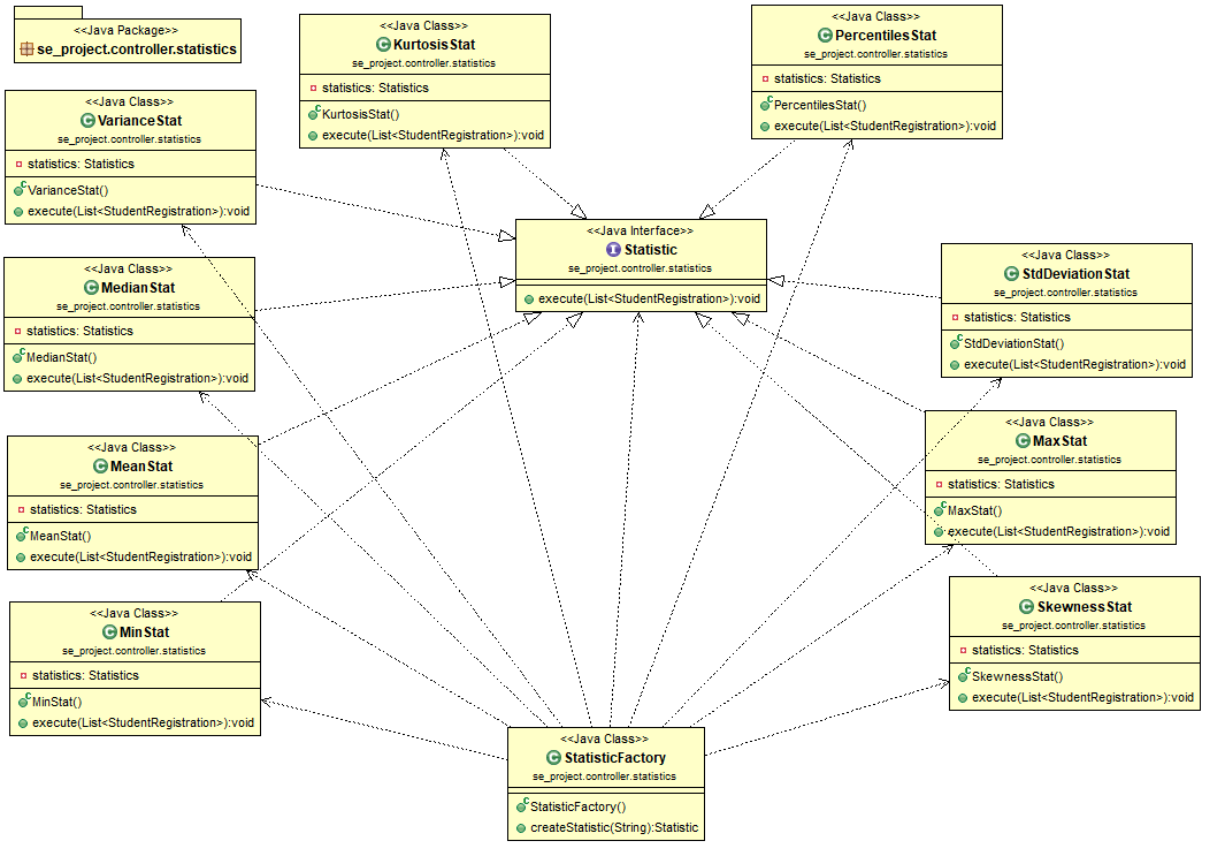
4.2 Design

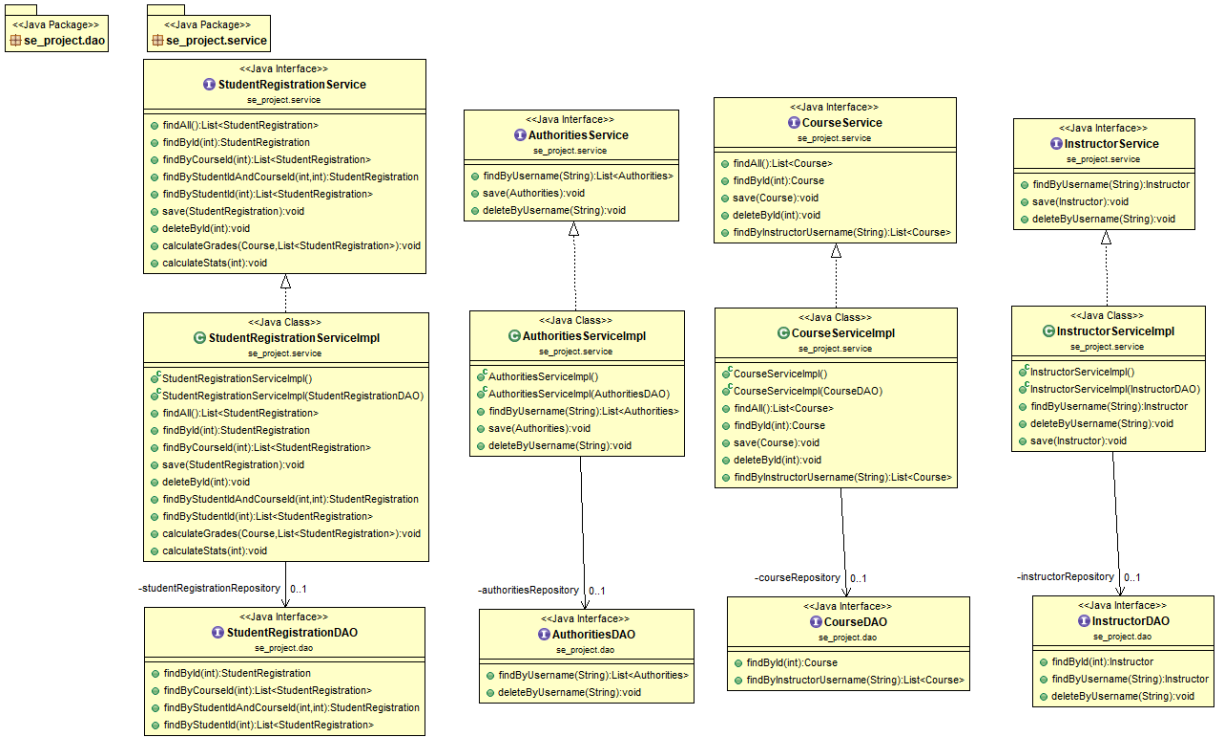


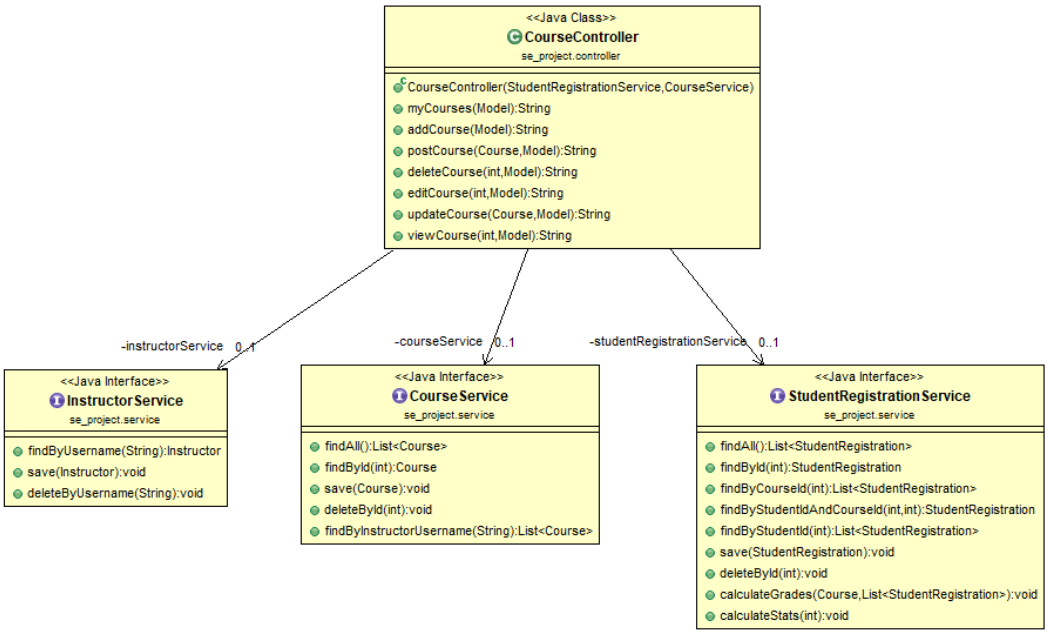


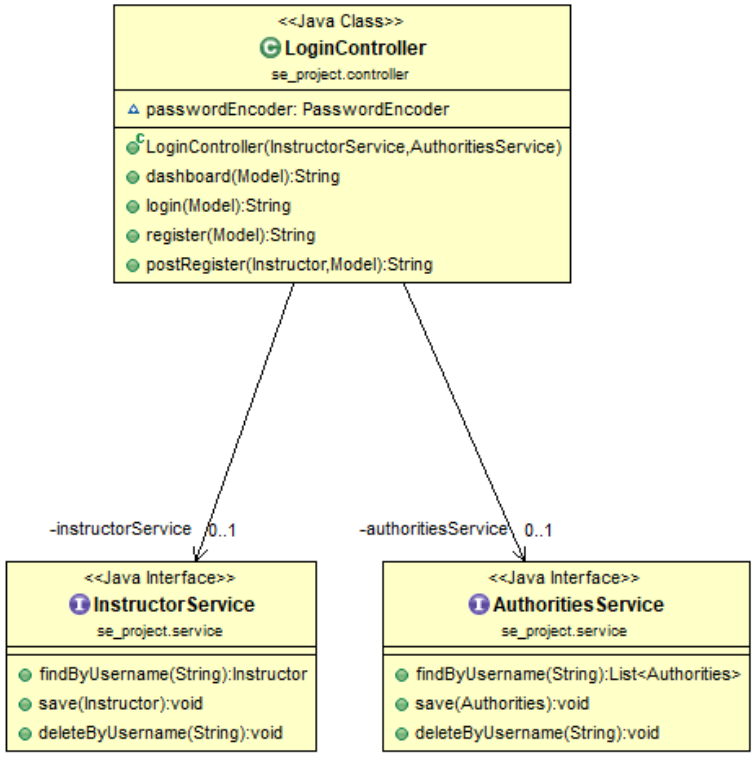

<<Java Package>>
se_project.entity
<<Java Class>>
StudentRegistration
se_project.entity
id: int
studentId: int
name: String
yearOfRegistration: int
semester: int
grade: double
projectGrade: double
examGrade: double
StudentRegistration()
StudentRegistration(int,String,int,int,Course,double,double,double)
getId():int
setId(int):void
getStudentId():int
setStudentId(int):void
getName():String
setName(String):void
getYearOfRegistration():int
setYearOfRegistration(int):void
getSemester():int
setSemester(int):void
getCourse():Course
setCourse(Course):void
getGrade():double
setGrade(double):void
getProjectGrade():double
setProjectGrade(double):void
getExamGrade():double
setExamGrade(double):void
toString():String
-course
0..1
<<Java Class>>
Course
se_project.entity
id: int
name: String
syllabus: String
year: int
semester: int
examWeight: double
projectWeight: double
Course()
Course(String,Instructor,String,int,int,double,double)
getId():int
setId(int):void
getName():String
setName(String):void
getInstructor():Instructor
setInstructor(Instructor):void
getSyllabus():String
setSyllabus(String):void
getYear():int
setYear(int):void
getSemester():int
setSemester(int):void
getExamWeight():double
setExamWeight(double):void
getProjectWeight():double
setProjectWeight(double):void
toString():String
-instructor
0..1
<<Java Class>>
Instructor
se_project.entity
id: int
fullname: String
username: String
password: String
enabled: boolean
Instructor()
Instructor(String,String,String,boolean)
getId():int
setId(int):void
getFullname():String
setFullname(String):void
getUsername():String
setUsername(String):void
getPassword():String
setPassword(String):void
isEnabled():boolean
setEnabled(boolean):void
toString():String
<<Java Class>>
Authorities
se_project.entity
username: String
authority: String
Authorities()
Authorities(String,String)
getUsername():String
setUsername(String):void
getAuthority():String
setAuthority(String):void
toString():String
<<Java Class>>
Statistics
se_project.entity
min: float
max: float
mean: float
stDeviation: float
variance: float
percentiles: HashMap<Integer,Double>
skewness: float
kurtosis: float
median: float
Statistics(float,float,float,float,float,HashMap<Integer,Double>,float,float,float)
Statistics()
getMin():float
setMin(float):void
getMax():float
setMax(float):void
getMean():float
setMean(float):void
getStDeviation():float
setStDeviation(float):void
getVariance():float
setVariance(float):void
getPercentiles():HashMap<Integer,Double>
setPercentiles(HashMap<Integer,Double>):void
getSkewness():float
setSkewness(float):void
getKurtosis():float
setKurtosis(float):void
getMedian():float
setMedian(float):void
toString():String

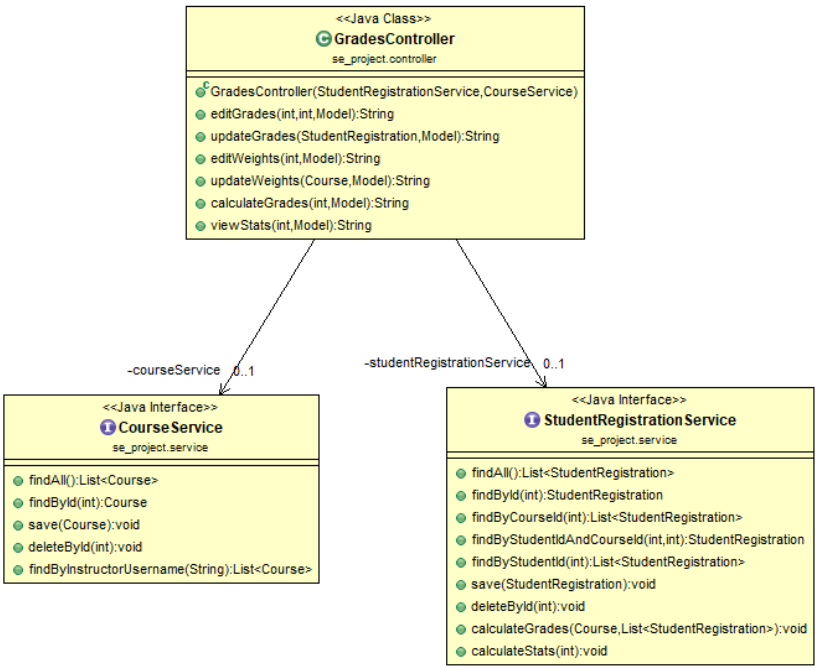

<<Java Package>>
se_project.controller
<<Java Class>>
CourseController
se_project.controller
courseService: CourseService
studentRegistrationService: StudentRegistrationService
instructorService: InstructorService
CourseController(StudentRegistrationService,CourseService)
myCourses(Model):String
addCourse(Model):String
postCourse(Course,Model):String
deleteCourse(int,Model):String
editCourse(int,Model):String
updateCourse(Course,Model):String
viewCourse(int,Model):String
<<Java Class>>
LoginController
se_project.controller
passwordEncoder: PasswordEncoder
instructorService: InstructorService
authoritiesService: AuthoritiesService
LoginController(InstructorService,AuthoritiesService)
dashboard(Model):String
login(Model):String
register(Model):String
postRegister(Instructor,Model):String
<<Java Class>>
GradesController
se_project.controller
courseService: CourseService
studentRegistrationService: StudentRegistrationService
GradesController(StudentRegistrationService,CourseService)
editGrades(int,int,Model):String
updateGrades(StudentRegistration,Model):String
editWeights(int,Model):String
updateWeights(Course,Model):String
calculateGrades(int,Model):String
viewStats(int,Model):String
<<Java Class>>
StudentController
se_project.controller
studentRegistrationService: StudentRegistrationService
courseService: CourseService
StudentController(StudentRegistrationService)
addStudent(int,Model):String
postStudent(StudentRegistration,Model):String
removeStudent(int,int,Model):String
editStudent(int,int,Model):String
updateStudent(StudentRegistration,String,Model):String
<<Java Class>>
Singleton
se_project.controller
statistics: Statistics
supportedStatistics: HashMap<String,Statistic>
Singleton()
getInstance():Singleton
-instance
0..1
<<Java Package>>
se_project.controller.statistics


<<Java Package>>
se_project.controller.statistics
<<Java Class>>
KurtosisStat
se_project.controller.statistics
statistics: Statistics
KurtosisStat()
execute(List<StudentRegistration>):void
<<Java Class>>
PercentilesStat
se_project.controller.statistics
statistics: Statistics
PercentilesStat()
execute(List<StudentRegistration>):void
<<Java Class>>
VarianceStat
se_project.controller.statistics
statistics: Statistics
VarianceStat()
execute(List<StudentRegistration>):void
<<Java Interface>>
Statistic
se_project.controller.statistics
execute(List<StudentRegistration>):void
<<Java Class>>
MedianStat
se_project.controller.statistics
statistics: Statistics
MedianStat()
execute(List<StudentRegistration>):void
<<Java Class>>
StdDeviationStat
se_project.controller.statistics
statistics: Statistics
StdDeviationStat()
execute(List<StudentRegistration>):void
<<Java Class>>
MeanStat
se_project.controller.statistics
statistics: Statistics
MeanStat()
execute(List<StudentRegistration>):void
<<Java Class>>
MaxStat
se_project.controller.statistics
statistics: Statistics
MaxStat()
execute(List<StudentRegistration>):void
<<Java Class>>
MinStat
se_project.controller.statistics
statistics: Statistics
MinStat()
execute(List<StudentRegistration>):void
<<Java Class>>
SkewnessStat
se_project.controller.statistics
statistics: Statistics
SkewnessStat()
execute(List<StudentRegistration>):void
<<Java Class>>
StatisticFactory
se_project.controller.statistics
StatisticFactory()
createStatistic(String):Statistic


<<Java Package>>
se_project.dao
<<Java Package>>
se_project.service
<<Java Interface>>
StudentRegistrationService
se_project.service
findAll():List<StudentRegistration>
findById(int):StudentRegistration
findByCourseId(int):List<StudentRegistration>
findByStudentIdAndCourseId(int,int):StudentRegistration
findByStudentId(int):List<StudentRegistration>
save(StudentRegistration):void
deleteById(int):void
calculateGrades(Course,List<StudentRegistration>):void
calculateStats(int):void
<<Java Interface>>
AuthoritiesService
se_project.service
findByUsername(String):List<Authorities>
save(Authorities):void
deleteByUsername(String):void
<<Java Interface>>
CourseService
se_project.service
findAll():List<Course>
findById(int):Course
save(Course):void
deleteById(int):void
findByInstructorUsername(String):List<Course>
<<Java Interface>>
InstructorService
se_project.service
findByUsername(String):Instructor
save(Instructor):void
deleteByUsername(String):void
<<Java Class>>
StudentRegistrationServiceImpl
se_project.service
StudentRegistrationServiceImpl()
StudentRegistrationServiceImpl(StudentRegistrationDAO)
findAll():List<StudentRegistration>
findById(int):StudentRegistration
findByCourseId(int):List<StudentRegistration>
save(StudentRegistration):void
deleteById(int):void
findByStudentIdAndCourseId(int,int):StudentRegistration
findByStudentId(int):List<StudentRegistration>
calculateGrades(Course,List<StudentRegistration>):void
calculateStats(int):void
<<Java Class>>
AuthoritiesServiceImpl
se_project.service
AuthoritiesServiceImpl()
AuthoritiesServiceImpl(AuthoritiesDAO)
findByUsername(String):List<Authorities>
save(Authorities):void
deleteByUsername(String):void
<<Java Class>>
CourseServiceImpl
se_project.service
CourseServiceImpl()
CourseServiceImpl(CourseDAO)
findAll():List<Course>
findById(int):Course
save(Course):void
deleteById(int):void
findByInstructorUsername(String):List<Course>
<<Java Class>>
InstructorServiceImpl
se_project.service
InstructorServiceImpl()
InstructorServiceImpl(InstructorDAO)
findByUsername(String):Instructor
deleteByUsername(String):void
save(Instructor):void
-studentRegistrationRepository
0..1
-authoritiesRepository
0..1
-courseRepository
0..1
-instructorRepository
0..1
<<Java Interface>>
StudentRegistrationDAO
se_project.dao
findById(int):StudentRegistration
findByCourseId(int):List<StudentRegistration>
findByStudentIdAndCourseId(int,int):StudentRegistration
findByStudentId(int):List<StudentRegistration>
<<Java Interface>>
AuthoritiesDAO
se_project.dao
findByUsername(String):List<Authorities>
deleteByUsername(String):void
<<Java Interface>>
CourseDAO
se_project.dao
findById(int):Course
findByInstructorUsername(String):List<Course>
<<Java Interface>>
InstructorDAO
se_project.dao
findById(int):Instructor
findByUsername(String):Instructor
deleteByUsername(String):void


<<Java Class>>
CourseController
se_project.controller
CourseController(StudentRegistrationService,CourseService)
myCourses(Model):String
addCourse(Model):String
postCourse(Course,Model):String
deleteCourse(int,Model):String
editCourse(int,Model):String
updateCourse(Course,Model):String
viewCourse(int,Model):String
-instructorService 0..1
-courseService 0..1
-studentRegistrationService 0..1
<<Java Interface>>
InstructorService
se_project.service
findByUsername(String):Instructor
save(Instructor):void
deleteByUsername(String):void
<<Java Interface>>
CourseService
se_project.service
findAll():List<Course>
findById(int):Course
save(Course):void
deleteById(int):void
findByInstructorUsername(String):List<Course>
<<Java Interface>>
StudentRegistrationService
se_project.service
findAll():List<StudentRegistration>
findById(int):StudentRegistration
findByCourseId(int):List<StudentRegistration>
findByStudentIdAndCourseId(int,int):StudentRegistration
findByStudentId(int):List<StudentRegistration>
save(StudentRegistration):void
deleteById(int):void
calculateGrades(Course,List<StudentRegistration>):void
calculateStats(int):void


<<Java Class>>
LoginController
se_project.controller
passwordEncoder: PasswordEncoder
LoginController(InstructorService,AuthoritiesService)
dashboard(Model):String
login(Model):String
register(Model):String
postRegister(Instructor,Model):String
-instructorService
0..1
-authoritiesService
0..1
<<Java Interface>>
InstructorService
se_project.service
findByUsername(String):Instructor
save(Instructor):void
deleteByUsername(String):void
<<Java Interface>>
AuthoritiesService
se_project.service
findByUsername(String):List<Authorities>
save(Authorities):void
deleteByUsername(String):void


<<Java Class>>
GradesController
se_project.controller
GradesController(StudentRegistrationService,CourseService)
editGrades(int,int,Model):String
updateGrades(StudentRegistration,Model):String
editWeights(int,Model):String
updateWeights(Course,Model):String
calculateGrades(int,Model):String
viewStats(int,Model):String
-courseService
0..1
-studentRegistrationService
0..1
<<Java Interface>>
CourseService
se_project.service
findAll():List<Course>
findById(int):Course
save(Course):void
deleteById(int):void
findByInstructorUsername(String):List<Course>
<<Java Interface>>
StudentRegistrationService
se_project.service
findAll():List<StudentRegistration>
findById(int):StudentRegistration
findByCourseId(int):List<StudentRegistration>
findByStudentIdAndCourseId(int,int):StudentRegistration
findByStudentId(int):List<StudentRegistration>
save(StudentRegistration):void
deleteById(int):void
calculateGrades(Course,List<StudentRegistration>):void
calculateStats(int):void

Class Name: Authorities	
Responsibilities: ▪ This class creates authority objects and holds their properties	**Collaborations:** ▪ This class is used by the Authorities DAO to store authorities and grab them from the database. ▪ This is also used by the security for the login.

Class Name: Course	
Responsibilities: ▪ This class creates course objects and holds their properties	**Collaborations:** ▪ This class is used by the Course DAO to store courses and grab them from the database. ▪ This is also used by the CourseController to implement the courses related user stories.

Class Name: Instructor	
Responsibilities: ▪ This class creates instructor objects and holds their properties	**Collaborations:** ▪ This class is used by the Instructor DAO to store instructors and grab them from the database. ▪ This is also used by the LoginController and the security for the login & registration.

Class Name: StudentRegistration	
Responsibilities: ▪ This class creates student registration objects and holds their properties	**Collaborations:** ▪ This class is used by the StudentRegistration DAO to store student registrations and grab them from the database. ▪ This is also used by the StudentController to implement the student related user stories.

Class Name: Statistics	
Responsibilities: ▪ This class creates Statistics objects and holds their properties	**Collaborations:** ▪ This class is used by the GradesController and the StudentRegistration services to calculate and view the statistics.

Class Name: Singleton	
Responsibilities: ▪ Holds shared fields used by multiple other classes (statistics)	**Collaborations:** ▪ Collaborates with all the statistics classes. ▪ It also collaborates with the StatisticFactory class, which is getting created on the Singleton class.

Class Name: StatisticFactory	
Responsibilities: ▪ Creates the supported statistics.	**Collaborations:** ▪ Collaborates with all the statistics classes. ▪ It gets executed from the singleton class to instantiate the statistics.