CoursesManagementApp

Sprint Report

Team Members :

Ilias Tsinoremas AM : 2151

Eirini Thoma AM : 4292

Nikolaos Vouronikos AM :4327

VERSIONS HISTORY

Date	Version	Description	Authors
13 May 2022	<1.4>	Final report for the course’s project	Nikolaos Vouronikos, Eirini Thoma Ilias Tsinoremas

1 Introduction

1.1 Purpose

This document provides information concerning the **<fourth>** sprint of the project. This is the final version of the report.

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

In the table below you can find the user stories included in this release along with their test methods.

User Story	Test Class	Test Method
2	**TestCoursesManagmentAppController**	**testDListCourses**
3	**TestCoursesManagmentAppController**	**testECourseFormAndAddSuccess**
4	**TestCoursesManagmentAppController**	**testGDeleteCourse**
5	**TestCoursesManagmentAppController**	**testFUpdateCourse**
1	**TestCoursesManagmentAppController**	**testCLoginPage**
Safe logout	**TestCoursesManagmentAppController**	**testLogoutPage**
6	**TestCoursesManagmentAppController**	**testIListStudentRegistrations**
7	**TestCoursesManagmentAppController**	**testJStudentFormAndSuccess**
8	**TestCoursesManagmentAppController**	**testNStudentDelete**
9	**TestCoursesManagmentAppController**	**testKStudentUpdate**

10	TestCoursesManagmentAppController	testLRegisterGradesPageSuccess
11	TestCoursesManagmentAppController	testMRegisterFinalGradeSuccess
12	TestCoursesManagmentAppController TestStatisticStrategy	

2.1 Scrum team

Product Owner	Eirini Thoma
Scrum Master	Nikolaos Vouronikos
Development Team	Eirini Thoma,Ilias Tsinoremas and Nikolaos Vouronikos

2.2 Sprints

Sprint No	Begin Date	End Date	Number of weeks	User stories
1	**18 March**	**2 April**	**2 weeks**	**US2,US3,US4 and US5**
2	**3 April**	**10 April**	**1 week**	**US1 and safe logout**
3	**10 April**	**17 April**	**1 week**	**US6 and US7**
4	**25 April**	**13 May**	**3 weeks**	**US8,US9,US10,US11 and US12**

3 Use Cases

3.1 <Use Case 1>

Use case ID	1
Actors	Instructor
Pre conditions	The instructor uses the right username and password.

Main flow of events	1. The use case starts when the user moves from home page to login page by clicking the 'click here' button. 2. User is prompted to write his username and password in the two text boxes respectively. 3. The instructor clicks the 'Sign In' button.
Alternative flow 1	If the user enters false username or password then he will be prompted to enter them again.
Post conditions	After the successful log in of the user,the application responds with a page containing his courses.

3.2 <Use Case 2>

Use case ID	2
Actors	Instructor
Pre conditions	The instructor hits the 'Add Course' button.
Main flow of events	1.The instructor is presented with a form containing the information of a course uncompleted. 2.Instructor enters all the information about the new course. 3.Instructor presses the 'Save' button. 4.The application responds with a confirmation page showing the information that the user entered. 5.The instructor clicks the 'Back to Course Directory' button and system repeats use case 3.
Alternative flow 1	1.The instructor enters invalid information. In this case the system shows an error page and prompts the user to go back to the previous page. 2.User clicks the 'Back to Course Directory' and cancels the course addition.
Post conditions	The list of the instructor's courses is updated and contains the new course.

3.3 <Use Case 3>

Use case ID	3
Actors	Instructor
Pre conditions	The instructor should have pressed 'Add Course' or 'Save' or 'Update' buttons.
Main flow of events	1.Instructor presses the 'Back to Course Directory' button.

Alternative flow 1	-
Post conditions	System shows the page containing the instructor's courses.

3.4 <Use Case 4>

Use case ID	4
Actors	Instructor
Pre conditions	The instructor clicks the 'Clear All' button.
Main flow of events	1.Application responds with an empty list without any courses.
Alternative flow 1	-
Post conditions	The list of the instructor's courses is empty.

3.5 <Use Case 5>

Use case ID	5
Actors	Instructor
Pre conditions	The instructor clicks the 'Sign Out' button.
Main flow of events	1.The system shows the login page. 2.System repeats use case 1.
Alternative flow 1	-
Post conditions	-

3.6 <Use Case 6>

Use case ID	6
Actors	Instructor
Pre conditions	The instructor clicks the 'Update' button.
Main flow of events	1.The system shows the form with all the information that the instructor has added for the course. 2.Instructor changes the information that he wants(ex.Period,Semester…). 3.Instructor clicks the 'Save' button. 4.The application responds with a confirmation page showing the information that the user entered.

	5.The instructor clicks the ‘Back to Course Directory’ button and system repeats use case 3.
Alternative flow 1	User clicks the ‘Back to Course Directory’ and cancels the course update.
Post conditions	System shows the instructor’s courses with the selected course’s information updated.

3.7 <Use Case 7>

Use case ID	7
Actors	Instructor
Pre conditions	The instructor clicks the ‘Delete’ button.
Main flow of events	1.The system responds with a dialog box. 2.Instructor presses the ‘yes’ button in the dialog box.
Alternative flow 1	If instructor presses the ‘no’ button in the dialog box the process is canceled.
Post conditions	The selected course has been deleted.

3.8 <Use Case 8>

Use case ID	8
Actors	Instructor
Pre conditions	The instructor clicks the ‘Students Info’ button.
Main flow of events	1.The system responds with a page containing all the students that are enrolled in the selected course.
Alternative flow 1	-
Post conditions	-

3.9 <Use Case 9>

Use case ID	9
Actors	Instructor
Pre conditions	The instructor hits the ‘Add Student‘ button.
Main flow of events	1.The instructor is presented with a form containing the information of a student uncompleted. 2.Instructor enters all the information about the new student.

	3.Instructor presses the ‘Save’ button. 4.The application responds with a confirmation page showing the information that the user entered. 5.The instructor clicks the ‘Back to Student Registrations’ button and system repeats use case 10.
Alternative flow 1	1.The instructor enters invalid information. In this case the system shows an error page and prompts the user to go back to the previous page. 2.User clicks the ‘Back to Student Registrations’ and cancels the student addition.
Post conditions	The list of the students that are enrolled in a particular course is updated with the new student.

3.10 <Use Case 10>

Use case ID	10
Actors	Instructor
Pre conditions	The instructor should have pressed ‘Add Student’ or ‘Save’ or ‘Update’ or ‘Delete’ or ‘Show All Grades’ or ‘Project Stats’ or ‘Exam Stats’ or ‘Final Grade Stats’ buttons.
Main flow of events	1.Instructor presses the ‘Back to Student Registrations’ button.
Alternative flow 1	-
Post conditions	The list containing all the students that are registered in the selected course.

3.11 <Use Case 11>

Use case ID	11
Actors	Instructor
Pre conditions	1.The instructor should have pressed ‘Project Stats’ or ‘Exam Stats’ or ‘Final Grade Stats’ buttons. 2.The instructor should have pressed either of the buttons in statistics menu.
Main flow of events	1.Instructor presses the ‘Back to Statistics Menu’ button.
Alternative flow 1	-
Post conditions	System shows the page containing the statistics menu.

3.12 <Use Case 12>

Use case ID	12
Actors	Instructor
Pre conditions	The instructor clicks the 'Show All Grades' button.
Main flow of events	-
Alternative flow 1	-
Post conditions	The instructor is presented with a list of all the students with their grades.

3.13 <Use Case 13>

Use case ID	13
Actors	Instructor
Pre conditions	The instructor clicks the 'Update' button.
Main flow of events	1.The system shows the form with all the information that the instructor has added for the student. 2.Instructor changes the information that he wants(ex.Registration Number,Student Name…). 3.Instructor clicks the 'Save' button. 4.The application responds with a confirmation page showing the information that the user entered. 5.The instructor clicks the 'Back to Student Registrations' button and system repeats use case 10.
Alternative flow 1	1.The instructor enters invalid information. In this case the system shows an error page and prompts the user to go back to the previous page. 2.User clicks the 'Back to Student Registrations' and cancels the student addition.
Post conditions	The list of the students that are enrolled in a particular course is updated with the the selected student's information updated.

3.14 <Use Case 14>

Use case ID	14
Actors	Instructor
Pre conditions	The instructor clicks the 'Delete' button.
Main flow of events	1.The system responds with a new page showing a confirmation message.

	2.Instructor presses the 'Back to Student Registrations' button. 3.System repeats use case 10.
Alternative flow 1	-
Post conditions	The selected student has been deleted.

3.15 <Use Case 15>

Use case ID	15
Actors	Instructor
Pre conditions	The instructor clicks the 'Calculate Final Grade' button.
Main flow of events	1.The system responds with a form containing two text boxes waiting from the user to insert the grades of the selected student(Project and exam grade respectively). 2.Instructor presses the 'Calculate Final Grade' button. 3.System responds with the information that the user has given along with the final grade of the student.
Alternative flow 1	The instructor enters invalid information. In this case the system shows an error page and prompts the user to go back to the previous page.
Post conditions	The selected student has his final grade been calculated.

3.16 <Use Case 16>

Use case ID	16
Actors	Instructor
Pre conditions	The instructor clicks the 'Project Stats' or 'Exam Stats' or 'Final Grade Stats' button.
Main flow of events	1.The System responds with a page containing all the utilities for statistics calculation.
Alternative flow 1	-
Post conditions	-

3.17 <Use Case 17>

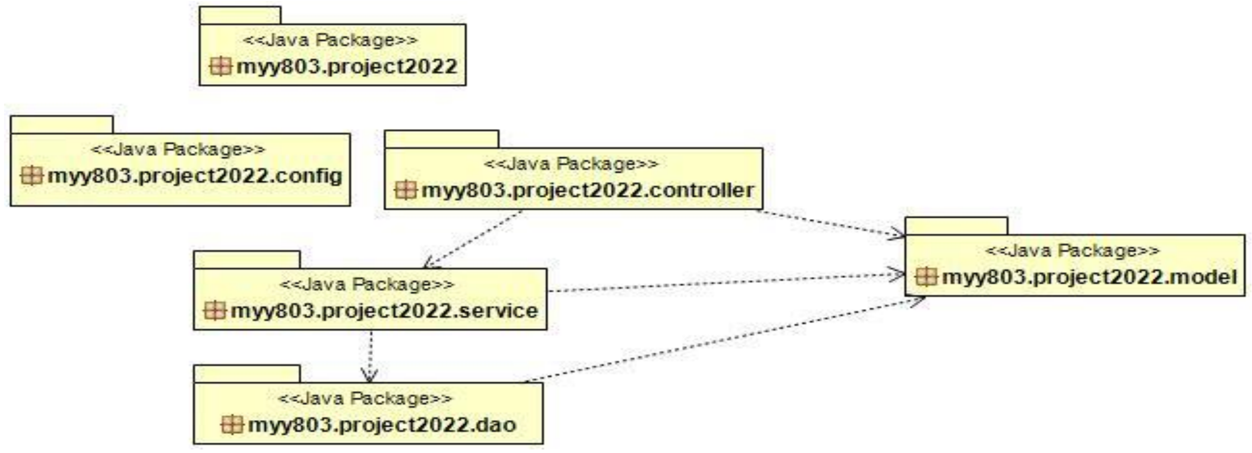
Use case ID	17

Actors	Instructor
Pre conditions	2.The instructor should have pressed either of the buttons in statistics menu.
Main flow of events	1.The System responds with the result that corresponds to the button that has been clicked.
Alternative flow 1	-
Post conditions	-

4 Design

4.1 Architecture

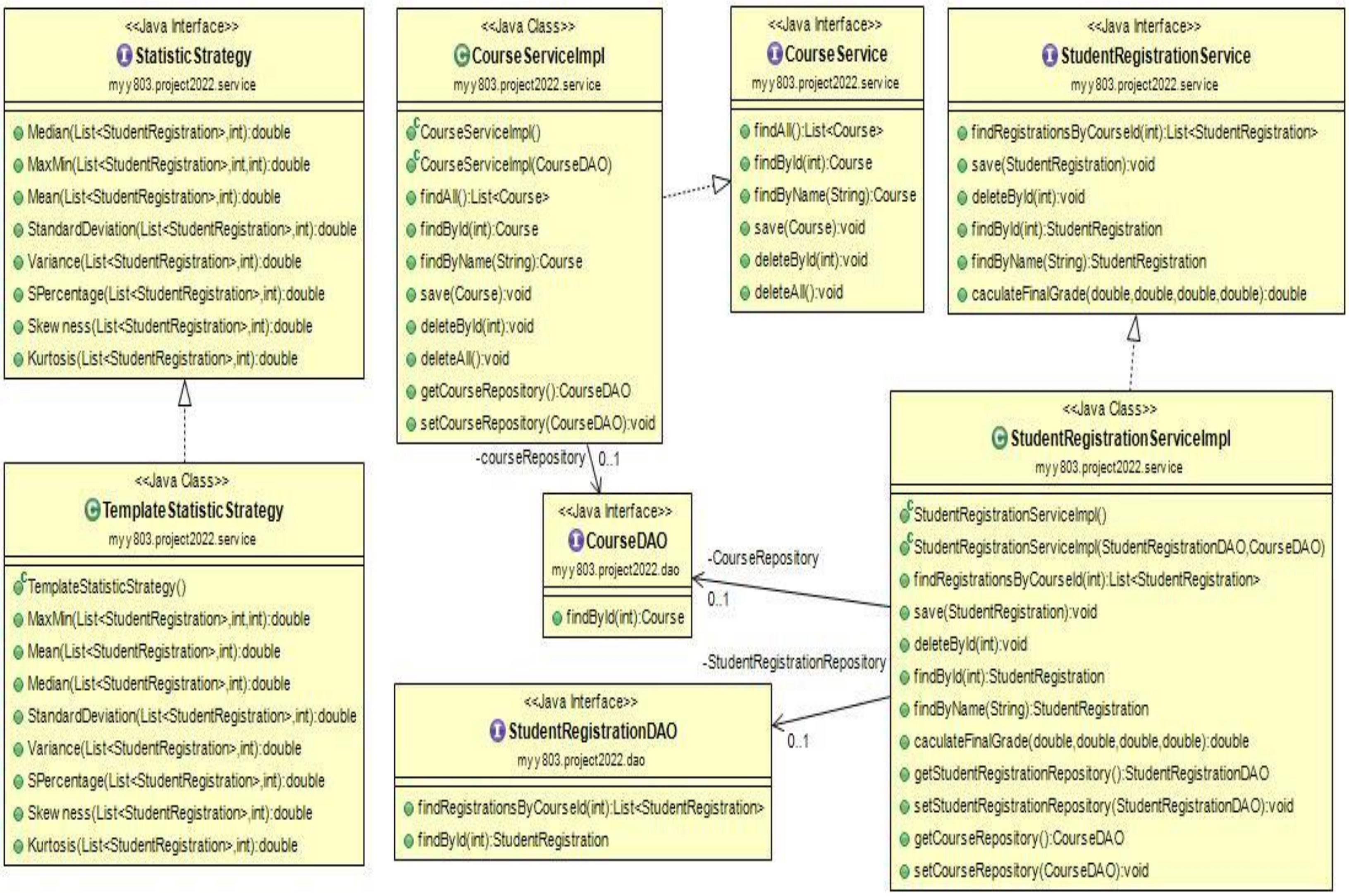
1.1) The image below shows the UML Package Diagram of our application :



4.2 Design

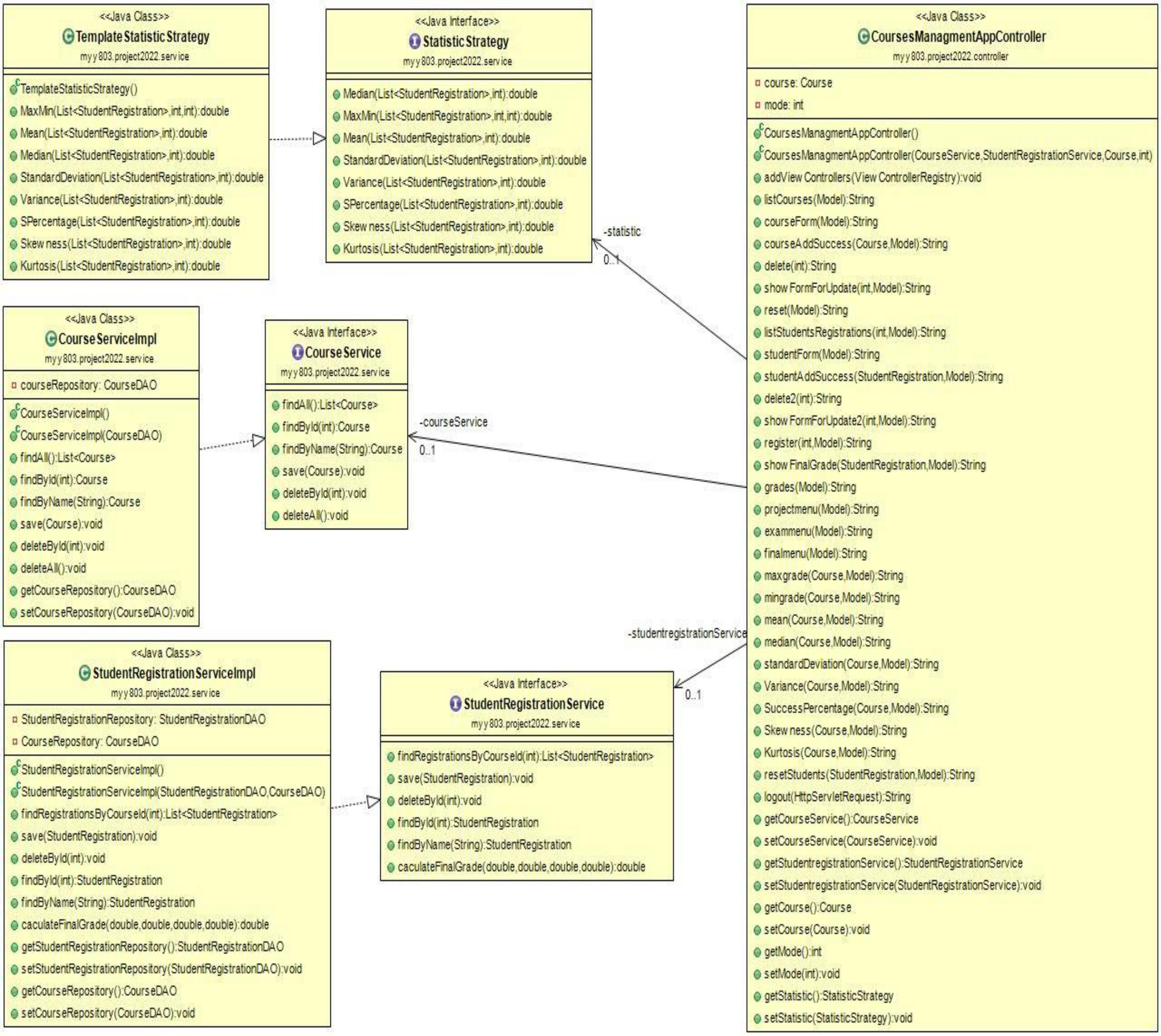
1.2) UML Class Diagram of our application(final):

Part 1 : Connection between the DAO layer classes and the Service layer classes.



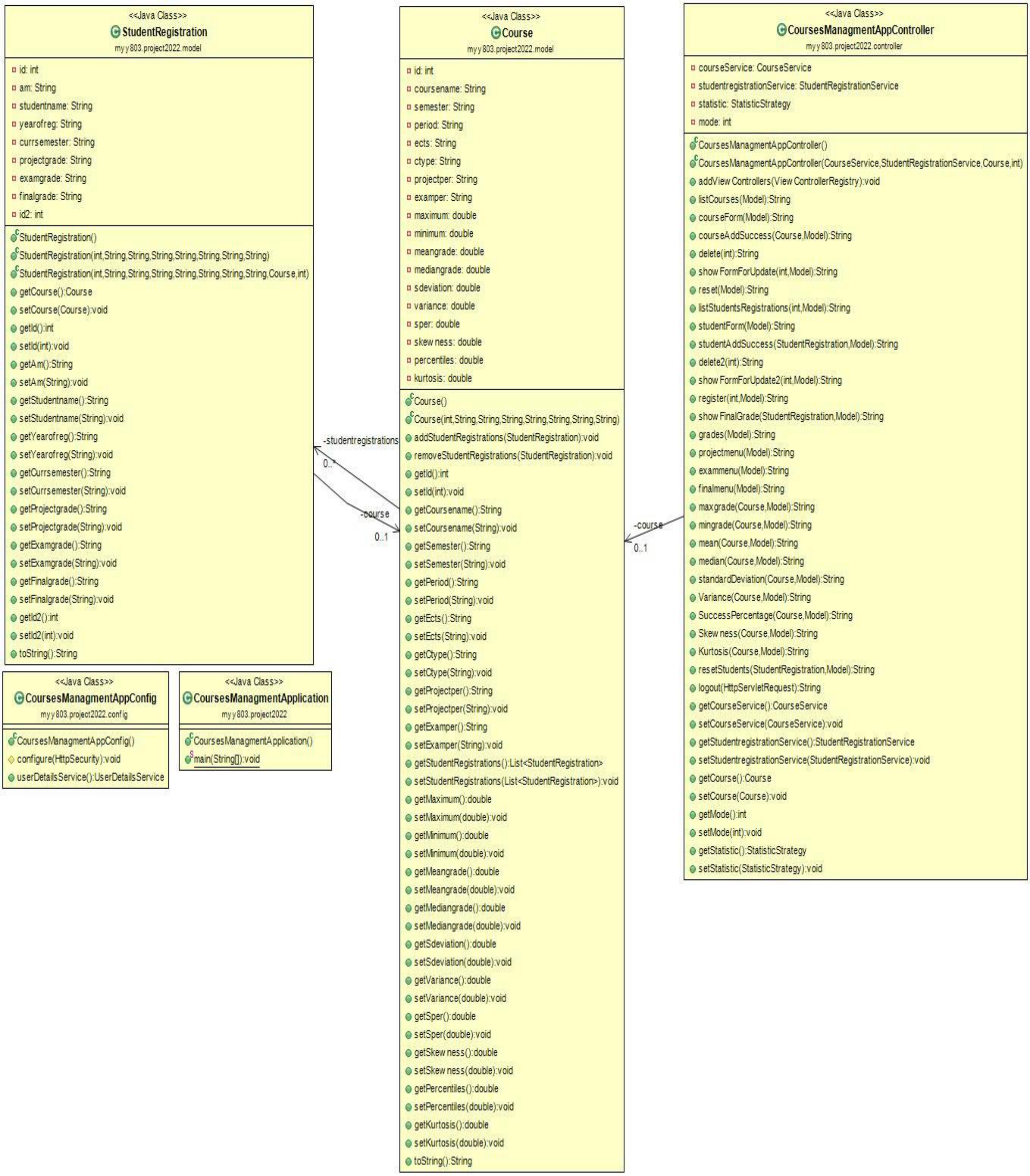
<<Java Interface>>
StatisticStrategy
myy803.project2022.service
Median(List<StudentRegistration>,int):double
MaxMin(List<StudentRegistration>,int,int):double
Mean(List<StudentRegistration>,int):double
StandardDeviation(List<StudentRegistration>,int):double
Variance(List<StudentRegistration>,int):double
SPercentage(List<StudentRegistration>,int):double
Skewness(List<StudentRegistration>,int):double
Kurtosis(List<StudentRegistration>,int):double
<<Java Class>>
CourseServiceImpl
myy803.project2022.service
CourseServiceImpl()
CourseServiceImpl(CourseDAO)
findAll():List<Course>
findById(int):Course
findByName(String):Course
save(Course):void
deleteById(int):void
deleteAll():void
getCourseRepository():CourseDAO
setCourseRepository(CourseDAO):void
<<Java Interface>>
CourseService
myy803.project2022.service
findAll():List<Course>
findById(int):Course
findByName(String):Course
save(Course):void
deleteById(int):void
deleteAll():void
<<Java Interface>>
StudentRegistrationService
myy803.project2022.service
findRegistrationsByCourseId(int):List<StudentRegistration>
save(StudentRegistration):void
deleteById(int):void
findById(int):StudentRegistration
findByName(String):StudentRegistration
caculateFinalGrade(double,double,double,double):double
-courseRepository
0..1
<<Java Class>>
TemplateStatisticStrategy
myy803.project2022.service
TemplateStatisticStrategy()
MaxMin(List<StudentRegistration>,int,int):double
Mean(List<StudentRegistration>,int):double
Median(List<StudentRegistration>,int):double
StandardDeviation(List<StudentRegistration>,int):double
Variance(List<StudentRegistration>,int):double
SPercentage(List<StudentRegistration>,int):double
Skewness(List<StudentRegistration>,int):double
Kurtosis(List<StudentRegistration>,int):double
<<Java Interface>>
CourseDAO
myy803.project2022.dao
findById(int):Course
-CourseRepository
0..1
-StudentRegistrationRepository
0..1
<<Java Interface>>
StudentRegistrationDAO
myy803.project2022.dao
findRegistrationsByCourseId(int):List<StudentRegistration>
findById(int):StudentRegistration
<<Java Class>>
StudentRegistrationServiceImpl
myy803.project2022.service
StudentRegistrationServiceImpl()
StudentRegistrationServiceImpl(StudentRegistrationDAO,CourseDAO)
findRegistrationsByCourseId(int):List<StudentRegistration>
save(StudentRegistration):void
deleteById(int):void
findById(int):StudentRegistration
findByName(String):StudentRegistration
caculateFinalGrade(double,double,double,double):double
getStudentRegistrationRepository():StudentRegistrationDAO
setStudentRegistrationRepository(StudentRegistrationDAO):void
getCourseRepository():CourseDAO
setCourseRepository(CourseDAO):void

Part 2 : Connection between the Service layer classes and the Controller layer class.



<<Java Class>>
TemplateStatisticStrategy
myy803.project2022.service
TemplateStatisticStrategy()
MaxMin(List<StudentRegistration>,int,int):double
Mean(List<StudentRegistration>,int):double
Median(List<StudentRegistration>,int):double
StandardDeviation(List<StudentRegistration>,int):double
Variance(List<StudentRegistration>,int):double
SPercentage(List<StudentRegistration>,int):double
Skewness(List<StudentRegistration>,int):double
Kurtosis(List<StudentRegistration>,int):double
<<Java Interface>>
StatisticStrategy
myy803.project2022.service
Median(List<StudentRegistration>,int):double
MaxMin(List<StudentRegistration>,int,int):double
Mean(List<StudentRegistration>,int):double
StandardDeviation(List<StudentRegistration>,int):double
Variance(List<StudentRegistration>,int):double
SPercentage(List<StudentRegistration>,int):double
Skewness(List<StudentRegistration>,int):double
Kurtosis(List<StudentRegistration>,int):double
-statistic
0..1
<<Java Class>>
CoursesManagmentAppController
myy803.project2022.controller
course: Course
mode: int
CoursesManagmentAppController()
CoursesManagmentAppController(CourseService,StudentRegistrationService,Course,int)
addViewControllers(ViewControllerRegistry):void
listCourses(Model):String
courseForm(Model):String
courseAddSuccess(Course,Model):String
delete(int):String
showFormForUpdate(int,Model):String
reset(Model):String
listStudentsRegistrations(int,Model):String
studentForm(Model):String
studentAddSuccess(StudentRegistration,Model):String
delete2(int):String
showFormForUpdate2(int,Model):String
register(int,Model):String
showFinalGrade(StudentRegistration,Model):String
grades(Model):String
projectmenu(Model):String
exammenu(Model):String
finalmenu(Model):String
maxgrade(Course,Model):String
mingrade(Course,Model):String
mean(Course,Model):String
median(Course,Model):String
standardDeviation(Course,Model):String
Variance(Course,Model):String
SuccessPercentage(Course,Model):String
Skewness(Course,Model):String
Kurtosis(Course,Model):String
resetStudents(StudentRegistration,Model):String
logout(HttpServletRequest):String
getCourseService():CourseService
setCourseService(CourseService):void
getStudentregistrationService():StudentRegistrationService
setStudentregistrationService(StudentRegistrationService):void
getCourse():Course
setCourse(Course):void
getMode():int
setMode(int):void
getStatistic():StatisticStrategy
setStatistic(StatisticStrategy):void
<<Java Class>>
CourseServiceImpl
myy803.project2022.service
courseRepository: CourseDAO
CourseServiceImpl()
CourseServiceImpl(CourseDAO)
findAll():List<Course>
findById(int):Course
findByName(String):Course
save(Course):void
deleteById(int):void
deleteAll():void
getCourseRepository():CourseDAO
setCourseRepository(CourseDAO):void
<<Java Interface>>
CourseService
myy803.project2022.service
findAll():List<Course>
findById(int):Course
findByName(String):Course
save(Course):void
deleteById(int):void
deleteAll():void
-courseService
0..1
-studentregistrationService
0..1
<<Java Class>>
StudentRegistrationServiceImpl
myy803.project2022.service
StudentRegistrationRepository: StudentRegistrationDAO
CourseRepository: CourseDAO
StudentRegistrationServiceImpl()
StudentRegistrationServiceImpl(StudentRegistrationDAO,CourseDAO)
findRegistrationsByCourseId(int):List<StudentRegistration>
save(StudentRegistration):void
deleteById(int):void
findById(int):StudentRegistration
findByName(String):StudentRegistration
caculateFinalGrade(double,double,double,double):double
getStudentRegistrationRepository():StudentRegistrationDAO
setStudentRegistrationRepository(StudentRegistrationDAO):void
getCourseRepository():CourseDAO
setCourseRepository(CourseDAO):void
<<Java Interface>>
StudentRegistrationService
myy803.project2022.service
findRegistrationsByCourseId(int):List<StudentRegistration>
save(StudentRegistration):void
deleteById(int):void
findById(int):StudentRegistration
findByName(String):StudentRegistration
caculateFinalGrade(double,double,double,double):double

Part 3 : Connection between the Controller layer class and the Domain Model classes.



<<Java Class>>
StudentRegistration
myy803.project2022.model
id: int
am: String
studentname: String
yearofreg: String
currsemester: String
projectgrade: String
examgrade: String
finalgrade: String
id2: int
StudentRegistration()
StudentRegistration(int,String,String,String,String,String,String,String)
StudentRegistration(int,String,String,String,String,String,String,String,Course,int)
getCourse():Course
setCourse(Course):void
getId():int
setId(int):void
getAm():String
setAm(String):void
getStudentname():String
setStudentname(String):void
getYearofreg():String
setYearofreg(String):void
getCurrsemester():String
setCurrsemester(String):void
getProjectgrade():String
setProjectgrade(String):void
getExamgrade():String
setExamgrade(String):void
getFinalgrade():String
setFinalgrade(String):void
getId2():int
setId2(int):void
toString():String
<<Java Class>>
Course
myy803.project2022.model
id: int
coursename: String
semester: String
period: String
ects: String
ctype: String
projectper: String
examper: String
maximum: double
minimum: double
meangrade: double
mediangrade: double
sdeviation: double
variance: double
sper: double
skewness: double
percentiles: double
kurtosis: double
Course()
Course(int,String,String,String,String,String,String,String)
addStudentRegistrations(StudentRegistration):void
removeStudentRegistrations(StudentRegistration):void
getId():int
setId(int):void
getCoursename():String
setCoursename(String):void
getSemester():String
setSemester(String):void
getPeriod():String
setPeriod(String):void
getEcts():String
setEcts(String):void
getCtype():String
setCtype(String):void
getProjectper():String
setProjectper(String):void
getExamper():String
setExamper(String):void
getStudentRegistrations():List<StudentRegistration>
setStudentRegistrations(List<StudentRegistration>):void
getMaximum():double
setMaximum(double):void
getMinimum():double
setMinimum(double):void
getMeangrade():double
setMeangrade(double):void
getMediangrade():double
setMediangrade(double):void
getSdeviation():double
setSdeviation(double):void
getVariance():double
setVariance(double):void
getSper():double
setSper(double):void
getSkewness():double
setSkewness(double):void
getPercentiles():double
setPercentiles(double):void
getKurtosis():double
setKurtosis(double):void
toString():String
<<Java Class>>
CoursesManagmentAppController
myy803.project2022.controller
courseService: CourseService
studentregistrationService: StudentRegistrationService
statistic: StatisticStrategy
mode: int
CoursesManagmentAppController()
CoursesManagmentAppController(CourseService,StudentRegistrationService,Course,int)
addViewControllers(ViewControllerRegistry):void
listCourses(Model):String
courseForm(Model):String
courseAddSuccess(Course,Model):String
delete(int):String
showFormForUpdate(int,Model):String
reset(Model):String
listStudentsRegistrations(int,Model):String
studentForm(Model):String
studentAddSuccess(StudentRegistration,Model):String
delete2(int):String
showFormForUpdate2(int,Model):String
register(int,Model):String
showFinalGrade(StudentRegistration,Model):String
grades(Model):String
projectmenu(Model):String
exammenu(Model):String
finalmenu(Model):String
maxgrade(Course,Model):String
mingrade(Course,Model):String
mean(Course,Model):String
median(Course,Model):String
standardDeviation(Course,Model):String
Variance(Course,Model):String
SuccessPercentage(Course,Model):String
Skewness(Course,Model):String
Kurtosis(Course,Model):String
resetStudents(StudentRegistration,Model):String
logout(HttpServletRequest):String
getCourseService():CourseService
setCourseService(CourseService):void
getStudentregistrationService():StudentRegistrationService
setStudentregistrationService(StudentRegistrationService):void
getCourse():Course
setCourse(Course):void
getMode():int
setMode(int):void
getStatistic():StatisticStrategy
setStatistic(StatisticStrategy):void
-studentregistrations
0..*
-course
0..1
-course
0..1
<<Java Class>>
CoursesManagmentAppConfig
myy803.project2022.config
CoursesManagmentAppConfig()
configure(HttpSecurity):void
userDetailsService():UserDetailsService
<<Java Class>>
CoursesManagmentApplication
myy803.project2022
CoursesManagmentApplication()
main(String[]):void

Important! In the DEMO-LINK.txt where you can find the link for the demo video of the application, in the same link you will find also the full image of the complete uml class diagram.

1.3) CRC Cards

Class Name: `CoursesManagmentAppController`	
Responsibilities:	**Collaborations:**
▪ Safe login and logout. ▪ Show the list of user's courses. ▪ Add a new course in the list. ▪ Update the information of an existing course. ▪ Delete a course. ▪ Delete all courses at once. ▪ Show the list of students that are enrolled in a particular course. ▪ Add a new student in the list. ▪ Update the information of an existing student. ▪ Delete a student. ▪ Delete all students at once. ▪ Show a table consisting of all students information along with their grades. ▪ Show statistics for a particular course.	▪ CourseService ▪ StudentRegistrationService ▪ StatisticStrategy ▪ Course ▪ StudentRegistration

Class Name: CourseService	
Responsibilities:	**Collaborations:**
▪ Find the list of user's courses. ▪ Find a course object using its id. ▪ Find a course object using its name. ▪ Save a course object into the database. ▪ Delete a course object, using its id,	▪ CourseDAO ▪ Course

from the database. ▪ Delete all course objects from the database.	

Class Name: StudentRegistrationService	
Responsibilities: ▪ Find all the student objects that are enrolled in a particular course using the course's id. ▪ Save a new StudentRegistration object in the database. ▪ Delete a StudentRegistration object using its id. ▪ Find a StudentRegistration object using its id. ▪ Fina a StudentRegistration object using its name. ▪ Calculate the final grade of a particular StudentRegistration object.	**Collaborations:** ▪ CourseDAO ▪ StudentRegistrationDAO ▪ Course ▪ StudentRegistration

Class Name: Course	
Responsibilities: ▪ Communicate with the database and the table course. ▪ Implement OneToMany relationship with the StudentRegistration objects.	**Collaborations:** ▪ StudentRegistration

Class Name: StudentRegistration	
Responsibilities: ▪ Communicate with the database and the table studentregistration. ▪ Implement ManyToOne relationship with a Course object.	**Collaborations:** ▪ Course

Class Name: StatisticStrategy	
Responsibilities: ▪ Calculates max value from a list of doubles. ▪ Calculates min value from a list of doubles. ▪ Calculates mean value from a list of doubles. ▪ Calculates median value from a list of doubles. ▪ Calculates standard deviation from a list of doubles. ▪ Calculates variance from a list of doubles. ▪ Calculates success percentage from a list of grades. ▪ Calculates skewness from a list of doubles. ▪ Calculates Kurtosis from a list of doubles.	**Collaborations:** ▪ Course