Courses Management App

Sprint Report

Μύρων Κουφόπουλος AM : 4398

Ανδρέας Τριανταφυλλόπουλος AM : 4504

VERSIONS HISTORY

Date	Version	Description	Author
17.3.2022	1	Creation of the database(db) (database_schema.sql).	
23.3.2022	2	Creation of demoApplication.java(main) , Course.java and data access object(DAO) files : CourseDAO.java.	
26.3.2022	3	Started with simple tests trying to add course objects to the db and then changing some variables so we can see if it has also been changed in the db. At last we tried the delete function of the DAO objects.	
01.4.2022	4	After additional testing and debugging, the connection between dao and our java program was established.	
02.4.2022	5	Creation of StudentRegistrationDAO.java and StudentRegistration.java with simple tests.	
05.4.2022	6	Add StudentRegistration objects inside every course and verify their existence in the db.	
08.4.2022	7	Creation of the CourseController.java , CourseService.java and CourseServiceImpl.java.	
13.4.2002	8	Creation of list-courses.html and testing if all courses are visible to the user.	
16.4.2022-18.4.2022	9	Then we added the ‘delete’,’update’ and ‘add course’ button with the proper html files.	
26.4.2022	10	Addition of showing the student registration button for every course and verify that every registration field is valid.	
28.4.2022	11	We added all the necessary html files that we needed to edit,create or delete any studentRegistrations.	
30.4.2022	12	Created the functions to calculate the statistics for every course and manufactured the html needed.	
01.5.2022	13	Production of DemoApplicationSecurity.java.	
3.5.2022-6.5.2022	14	Production of junit tests.	
7.5.2022-8.5.2022	15	Debugg and clean the code.	

1 Introduction

1.1 Purpose

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

<For the user stories included in this release specify below corresponding tests using a typical tabular form.>

2.1 Scrum team

Product Owner	Myron Koufopoulos, Andreas Triantafyllopoulos
Scrum Master	
Development Team	

2.2 Sprints

Sprint No	Begin Date	End Date	Number of days	User stories
1	1/5/2022	1/5/2022	1	**US1:** login to the application with a user name and a password.
2	13/4/2022	14/4/2022	2	**US2:** browse the list of the courses.
3	16/4/2022	18/4/2022	3	**US3:** add a course in the list, by giving information : the course id, name , syllabus, semester, project percentage and exams percentage.

4	16/4/2022	18/4/2022	3	**US4:** remove a course from the list.
5	16/4/2022	18/4/2022	3	**US5:** update the description of a course.
6	26/4/2022	27/4/2022	2	**US6:** browse the list of students that enrolled to a particular course.
7	28/4/2022	29/4/2022	2	**US7:** add a student to the list of a particular course and giving the student name , year of studies, semester, project grade and exams grade (the final grade will be calculated autonomously).
8	28/4/2022	29/4/2022	2	**US8:** remove a student from a list of particular course.
9	28/4/2022	29/4/2022	2	**US9:** update for every student the field we want to change.
10	28/4/2022	29/4/2022	2	**US10:** register the grades of the student in the exams and the project of the course.
11	30/4/2022	1/5/2022	2	**US11:** calculate the overall final grades of the students that enrolled in a course taking into consider the weighted average.
12	30/4/2022	1/5/2022	2	**US12:** calculate the statistics for every course about the student grades (average project grade, average exams grade, average finals grade, min final grade, max final grade, mean final grade, median final grade)

3 Use Cases

3.1 Login

Use case ID	1
Actors	user
Pre conditions	The db and the server are both active.
Main flow of events	[Main flow of events that describes the interaction between the user and the application] 1. The use case starts when the user send any kind of request on the local host over the port 8080 2. The system asks for credentials to continue (login and password) 3. The user fills in the necessary fields
Alternative flow 1	If the user enter wrong username or wrong password the website informs him that he can't log in and asks to re-enter them.
Post conditions	The system continues by displaying the list of courses

3.2 BrowseCourses

Use case ID	2
Actors	user
Pre conditions	The user has log on to the website.
Main flow of events	1. The use case starts when the user has requested to see the list of courses (which happens automatically after the user has log on) 2. The system shows the list of courses
Alternative flow 1	If the user is currently seeing the StudentRegistrations of a course and requests to go back and see the list of courses.
Alternative flow 2	If the user is currently seeing the statistics of a course, the update or add course form and requests to go back and see the list of courses.
Post conditions	User sees all the courses that are available

3.3 AddCourse

Use case ID	3
Actors	user
Pre conditions	The user is able to see the list of courses.
Main flow of events	1. The use case starts when the user has requested to add a course 2. The system shows an empty form to fill in with all the course specifics 3. The user fills in the form 4. The system saves the new course in the db
Post conditions	User sees the list of courses

3.4 DeleteCourse

Use case ID	4
Actors	user
Pre conditions	The user is able to see the list of courses and the db has already at least one course.
Main flow of events	1. The use case starts when the user has requested to delete a course 2. The system deletes the course from the db
Post conditions	User sees the list of courses

3.5 UpdateCourse

Use case ID	5
Actors	user
Pre conditions	The user is able to see the list of courses and the db has already at least one course.
Main flow of events	1. The use case starts when the user has requested to update a course 2. The system shows the form of this exact course filled with the parameters it already has 3. The user updates the field he wants 4. The system saves the updated course in the db

Alternative flow 1	The user changes none of the courses parameters, instead he presses back or he saves it with the same prameters
Post conditions	User sees the list of courses

3.6 BrowseStudentRegistrations

Use case ID	6
Actors	user
Pre conditions	The user is able to see the list of courses and the db has already at least one course.
Main flow of events	1. The use case starts when the user has requested to browse the student registrations of a specific course 2. The system shows the list of student registrations for this course
Alternative flow 1	If the user is currently seeing the update or is adding a new student registration and requests to go back and see the list of student registrations.
Post conditions	User sees all the student registrations of the specific course that are available

3.7 AddStudentRegistrations

Use case ID	7
Actors	user
Pre conditions	The user is able to see the list of courses .
Main flow of events	1. The use case starts when the user has requested to add a student registrations of a specific course 2. The system shows an empty form to fill in with all the student registration specifics 3. The user fills in the form 4. The system saves the new student registration in the db
Post conditions	User sees all the student registrations of the specific course that are available

3.8 RemoveStudentRegistrations

Use case ID	8
Actors	user
Pre conditions	The user is able to see the list of courses and the db has already at least one student registration.
Main flow of events	1. The use case starts when the user has requested to delete a student registration 2. The system deletes the student registration from the db
Post conditions	User sees all the student registrations of the specific course that are available

3.9 UpdateStudentRegistration

Use case ID	9
Actors	user
Pre conditions	The user is able to see the list of courses and the db has already at least one student registration.
Main flow of events	1. The use case starts when the user has requested to update a student registration 2. The system shows the form of this exact student registration filled with the parameters it already has 3. The user updates the field he wants 4. The system saves the updated student registration in the db
Alternative flow 1	The user changes none of the student registration parameters, instead he presses back or he saves it with the same prameters
Post conditions	User sees all the student registrations of the specific course that are available

3.10 RegisterGrades

Use case ID	10
Actors	user
Pre conditions	The user is able to see the list of student registration of a specific course
Main flow of events	1. The use case starts when the user has requested to update a student registration

	2. The system shows the form of this exact student registration filled with the parameters it already has 3. The user fills in the exams and the project grade of the student 4. The System shows the final grade of the student 5. The user presses the save button and 6. The system adds the changes to the db
Post conditions	User sees all the student registrations of the specific course that are available

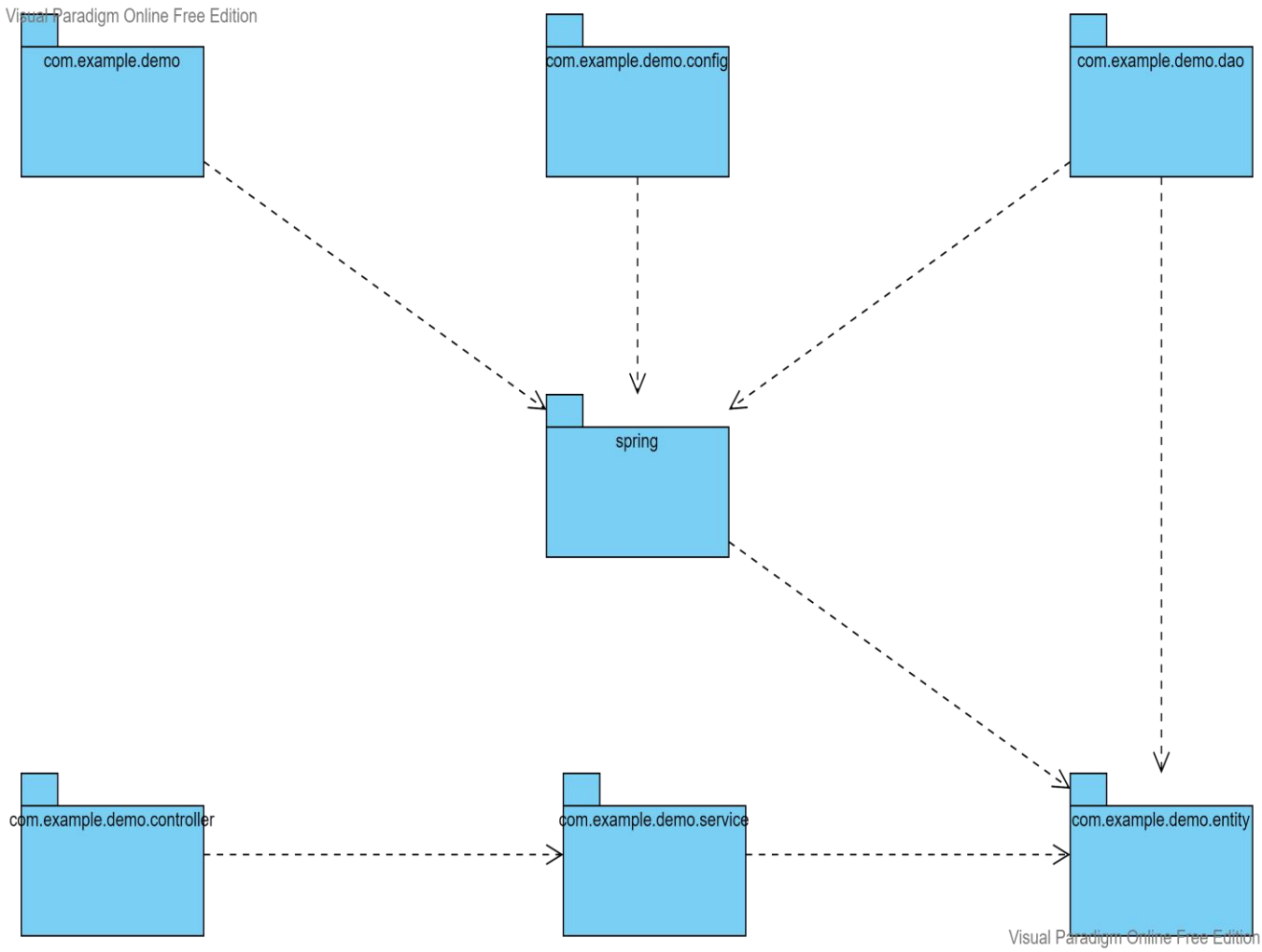
3.11 CalculateStats

Use case ID	11
Actors	user
Pre conditions	The user is able to see the list of courses and the db has already at least one course with at least one student registration.
Main flow of events	1. The use case starts when the user has requested to see the statistics of a specific course. 2. The system displays the calculated statistics. These are: a) The average project grade b) The average exams grade c) The average final grade d) The minimum final grade e) The maximum final grade f) The mean final grade g) The median final grade 3. The user presses the back button.
Post conditions	User sees the list of all courses

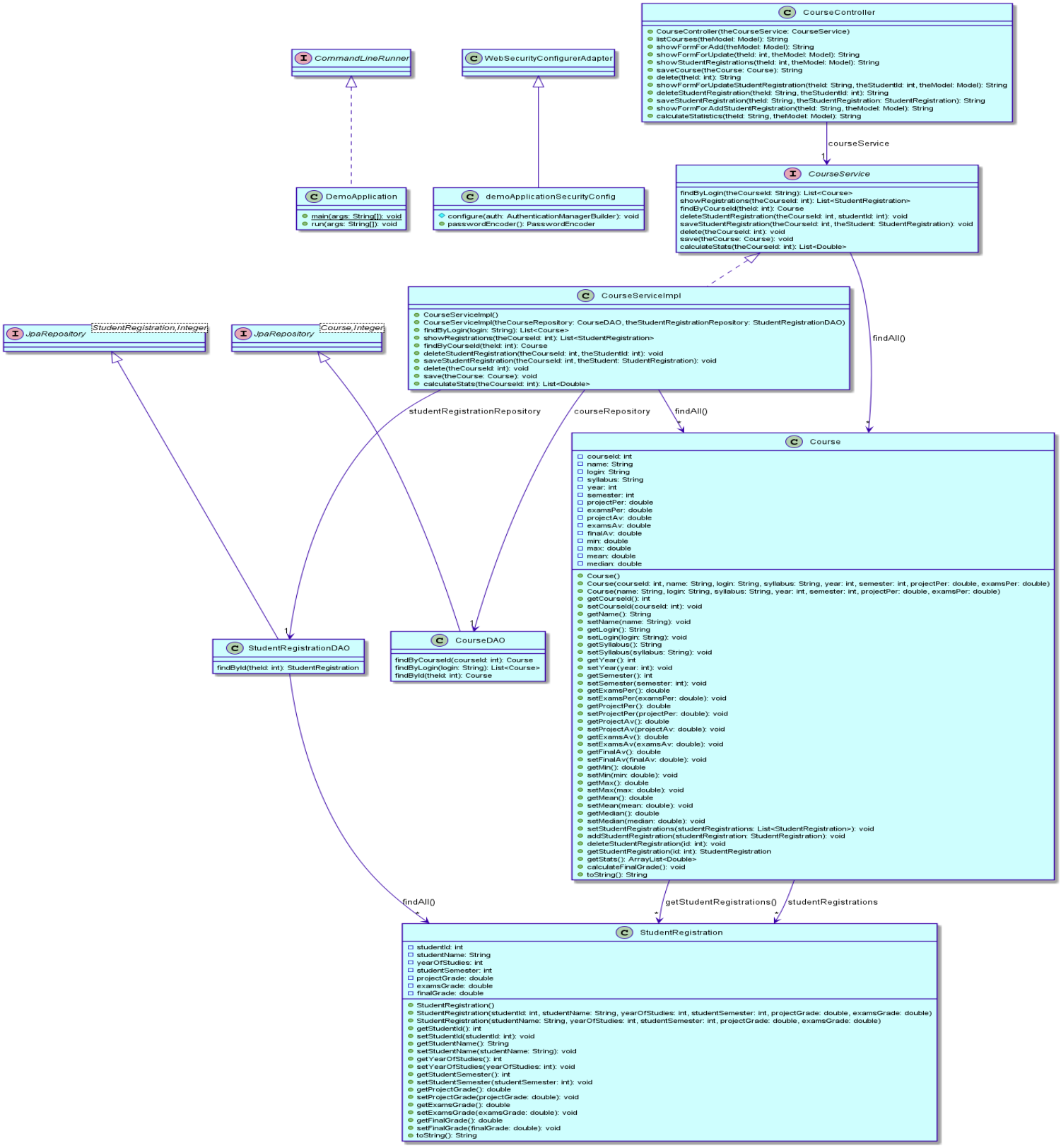
4 Design

4.1 Architecture

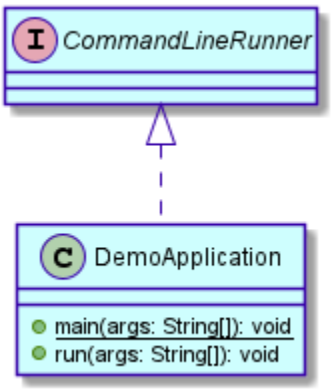
Below we present the package uml diagram of our software. We need to clarify that there is no “spring” package but there is functionality on some packages that go through the spring framework and we wanted to demonstrate it (i.e. the “dao” package saves data to the database through the spring framework etc)



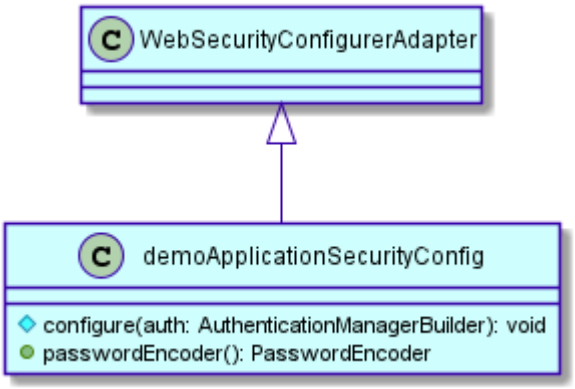
4.2 Design


CommandLineRunner
WebSecurityConfigurerAdapter
CourseController
CourseController(theCourseService: CourseService)
listCourses(theModel: Model): String
showFormForAdd(theModel: Model): String
showFormForUpdate(theId: int, theModel: Model): String
showStudentRegistrations(theId: int, theModel: Model): String
saveCourse(theCourse: Course): String
delete(theId: int): String
showFormForUpdateStudentRegistration(theId: String, theStudentId: int, theModel: Model): String
deleteStudentRegistration(theId: String, theStudentId: int): String
saveStudentRegistration(theId: String, theStudentRegistration: StudentRegistration): String
showFormForAddStudentRegistration(theId: String, theModel: Model): String
calculateStatistics(theId: String, theModel: Model): String
courseService
1
DemoApplication
main(args: String[]): void
run(args: String[]): void
demoApplicationSecurityConfig
configure(auth: AuthenticationManagerBuilder): void
passwordEncoder(): PasswordEncoder
CourseService
findByLogin(theCourseId: String): List<Course>
showRegistrations(theCourseId: int): List<StudentRegistration>
findByCourseId(theId: int): Course
deleteStudentRegistration(theCourseId: int, studentId: int): void
saveStudentRegistration(theCourseId: int, theStudent: StudentRegistration): void
delete(theCourseId: int): void
save(theCourse: Course): void
calculateStats(theCourseId: int): List<Double>
CourseServiceImpl
CourseServiceImpl()
CourseServiceImpl(theCourseRepository: CourseDAO, theStudentRegistrationRepository: StudentRegistrationDAO)
findByLogin(login: String): List<Course>
showRegistrations(theCourseId: int): List<StudentRegistration>
findByCourseId(theId: int): Course
deleteStudentRegistration(theCourseId: int, theStudentId: int): void
saveStudentRegistration(theCourseId: int, theStudent: StudentRegistration): void
delete(theCourseId: int): void
save(theCourse: Course): void
calculateStats(theCourseId: int): List<Double>
findAll()
JpaRepository
StudentRegistration,Integer
JpaRepository
Course,Integer
studentRegistrationRepository
courseRepository
findAll()
Course
courseId: int
name: String
login: String
syllabus: String
year: int
semester: int
projectPer: double
examsPer: double
projectAv: double
examsAv: double
finalAv: double
min: double
max: double
mean: double
median: double
Course()
Course(courseId: int, name: String, login: String, syllabus: String, year: int, semester: int, projectPer: double, examsPer: double)
Course(name: String, login: String, syllabus: String, year: int, semester: int, projectPer: double, examsPer: double)
getCourseId(): int
setCourseId(courseId: int): void
getName(): String
setName(name: String): void
getLogin(): String
setLogin(login: String): void
getSyllabus(): String
setSyllabus(syllabus: String): void
getYear(): int
setYear(year: int): void
getSemester(): int
setSemester(semester: int): void
getExamsPer(): double
setExamsPer(examsPer: double): void
getProjectPer(): double
setProjectPer(projectPer: double): void
getProjectAv(): double
setProjectAv(projectAv: double): void
getExamsAv(): double
setExamsAv(examsAv: double): void
getFinalAv(): double
setFinalAv(finalAv: double): void
getMin(): double
setMin(min: double): void
getMax(): double
setMax(max: double): void
getMean(): double
setMean(mean: double): void
getMedian(): double
setMedian(median: double): void
setStudentRegistrations(studentRegistrations: List<StudentRegistration>): void
addStudentRegistration(studentRegistration: StudentRegistration): void
deleteStudentRegistration(id: int): void
getStudentRegistration(id: int): StudentRegistration
getStats(): ArrayList<Double>
calculateFinalGrade(): void
toString(): String
StudentRegistrationDAO
findById(theId: int): StudentRegistration
CourseDAO
findByCourseId(courseId: int): Course
findByLogin(login: String): List<Course>
findById(theId: int): Course
findAll()
getStudentRegistrations()
studentRegistrations
StudentRegistration
studentId: int
studentName: String
yearOfStudies: int
studentSemester: int
projectGrade: double
examsGrade: double
finalGrade: double
StudentRegistration()
StudentRegistration(studentId: int, studentName: String, yearOfStudies: int, studentSemester: int, projectGrade: double, examsGrade: double)
StudentRegistration(studentName: String, yearOfStudies: int, studentSemester: int, projectGrade: double, examsGrade: double)
getStudentId(): int
setStudentId(studentId: int): void
getStudentName(): String
setStudentName(studentName: String): void
getYearOfStudies(): int
setYearOfStudies(yearOfStudies: int): void
getStudentSemester(): int
setStudentSemester(studentSemester: int): void
getProjectGrade(): double
setProjectGrade(projectGrade: double): void
getExamsGrade(): double
setExamsGrade(examsGrade: double): void
getFinalGrade(): double
setFinalGrade(finalGrade: double): void
toString(): String

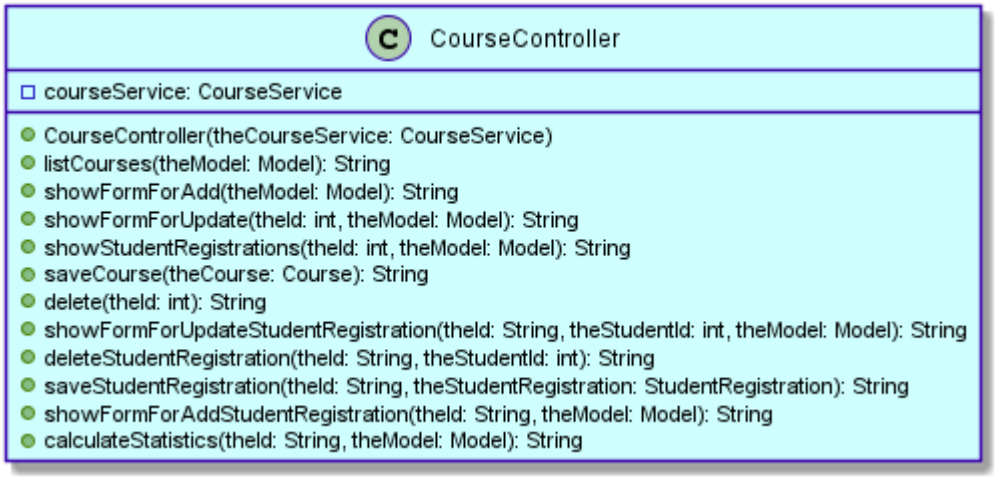
Class Name: DemoApplication	
Responsibilities: ▪ Run the program	**Collaborations:** ▪ Spring framework



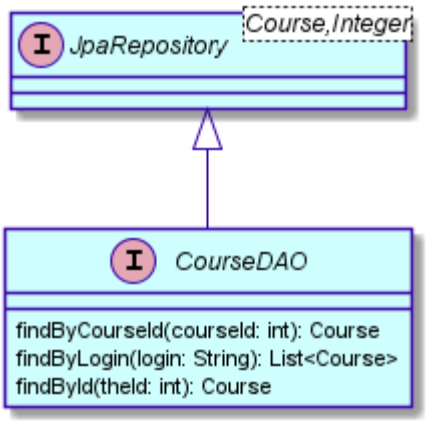
Class Name: DemoApplicationSecurityConfig	
Responsibilities: ▪ Keep the accounts safe ▪ Keep the passwords encrypted ▪ Keep the professor's username ▪ Specify the role for every user	**Collaborations:** ▪ Spring framework



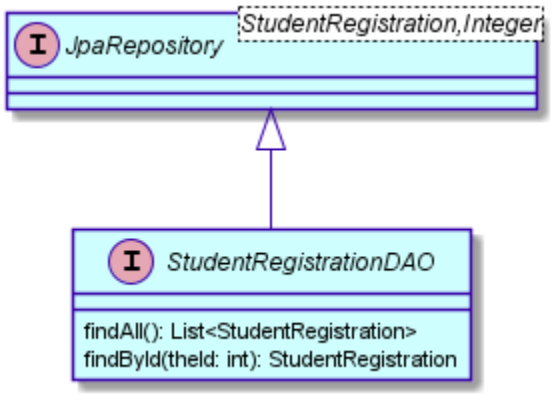
Class Name: CourseController	
Responsibilities: ▪ Manage http requests from the user (list-courses, showFormForUpdate, studentRegistrations, showFormForUpdate-studentRegistration) ▪ Instruct the db to save a new object ▪ Instruct the db to delete an object ▪ Instruct the db to update an object ▪ Request the statistics ▪ Redirect the user after an action ▪ Control which http will be displayed to the user	**Collaborations:** ▪ CourseServiceImpl



Class Name: CourseDAO	
Responsibilities: ▪ Manage the requests for the courses from the software to the db	**Collaborations:** ▪ Spring framework ▪ Mysql db



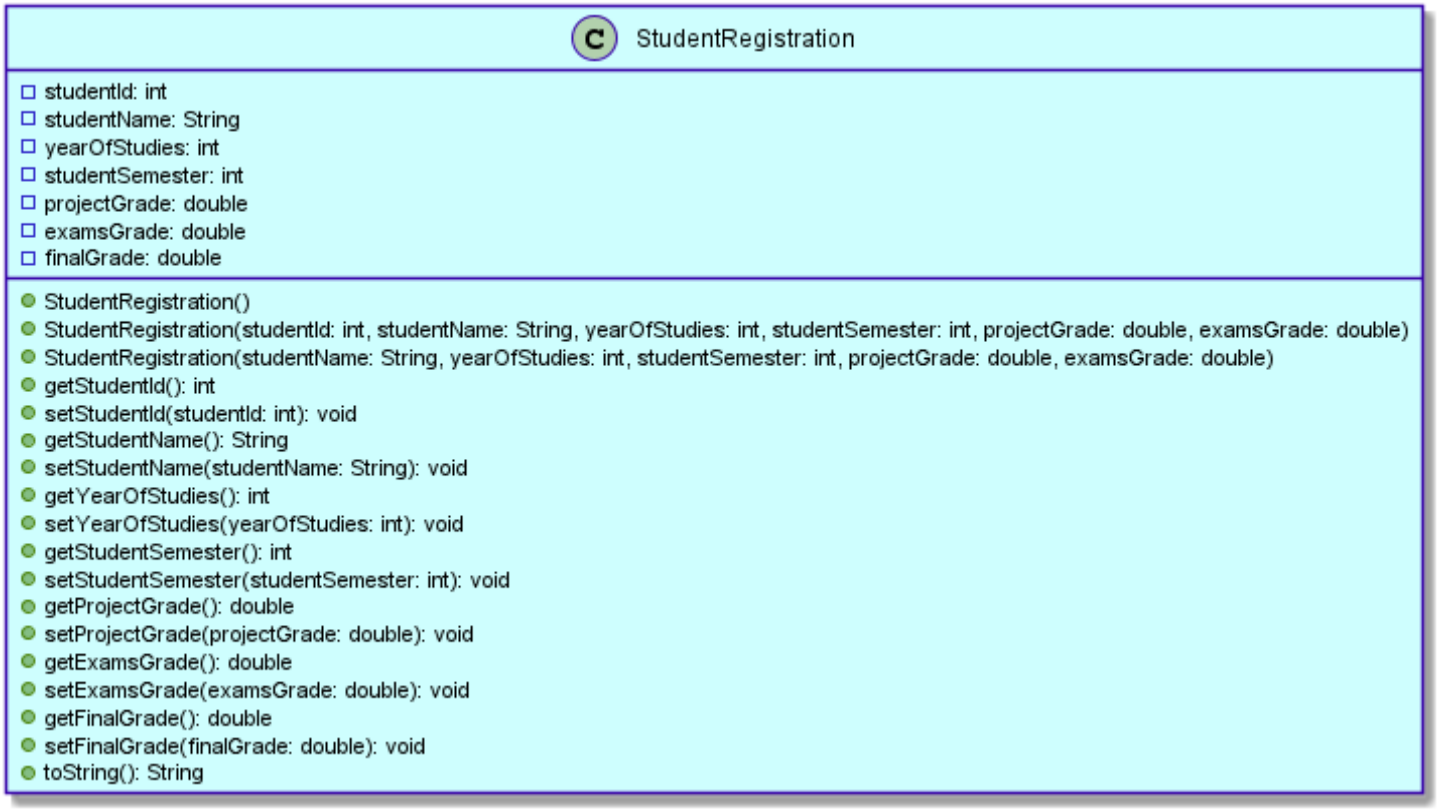
Class Name: StudentRegistrationDAO	
Responsibilities: ▪ Manage the requests for the student registrations from the software to the db	**Collaborations:** ▪ Spring framework ▪ Mysql db



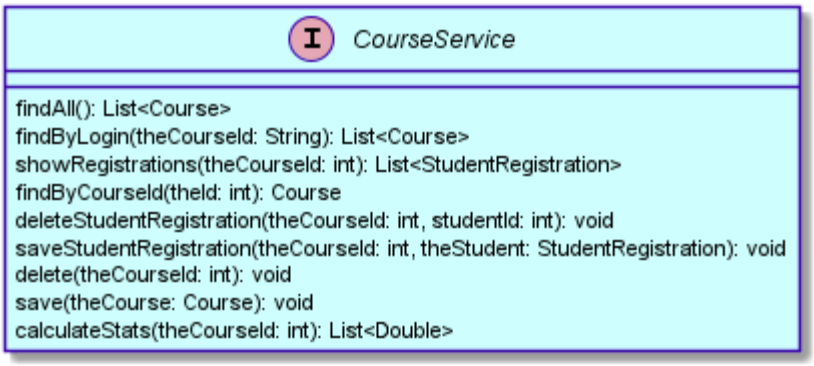
Class Name: Course	
Responsibilities: ▪ Inherit the fields of the course template from the db ▪ Manage all the fields that are needed for every course(courseId,name,login,…etc) ▪ Keep all the student registrations of a specific course (into an arraylist) ▪ Add student registrations to the db ▪ Delete student registration from the db ▪ Retrieve the student registrations that are related to a specific course ▪ Calculate the final grade for every student registration ▪ Calculate the statistics for every course a) avgProjectGrade b) avgExamGrade c) avgFinalGrade d) min e) max f) mean g) median	**Collaborations:** ▪ Mysql db


Course
courseId: int
name: String
login: String
syllabus: String
year: int
semester: int
projectPer: double
examsPer: double
projectAv: double
examsAv: double
finalAv: double
min: double
max: double
mean: double
median: double
studentRegistrations: List<StudentRegistration>
Course()
Course(courseId: int, name: String, login: String, syllabus: String, year: int, semester: int, projectPer: double, examsPer: double)
Course(name: String, login: String, syllabus: String, year: int, semester: int, projectPer: double, examsPer: double)
getCourseId(): int
setCourseId(courseId: int): void
getName(): String
setName(name: String): void
getLogin(): String
setLogin(login: String): void
getSyllabus(): String
setSyllabus(syllabus: String): void
getYear(): int
setYear(year: int): void
getSemester(): int
setSemester(semester: int): void
getExamsPer(): double
setExamsPer(examsPer: double): void
getProjectPer(): double
setProjectPer(projectPer: double): void
getProjectAv(): double
setProjectAv(projectAv: double): void
getExamsAv(): double
setExamsAv(examsAv: double): void
getFinalAv(): double
setFinalAv(finalAv: double): void
getMin(): double
setMin(min: double): void
getMax(): double
setMax(max: double): void
getMean(): double
setMean(mean: double): void
getMedian(): double
setMedian(median: double): void
getStudentRegistrations(): List<StudentRegistration>
setStudentRegistrations(studentRegistrations: List<StudentRegistration>): void
addStudentRegistration(studentRegistration: StudentRegistration): void
deleteStudentRegistration(id: int): void
getStudentRegistration(id: int): StudentRegistration
getStats(): ArrayList<Double>
calculateFinalGrade(): void
toString(): String

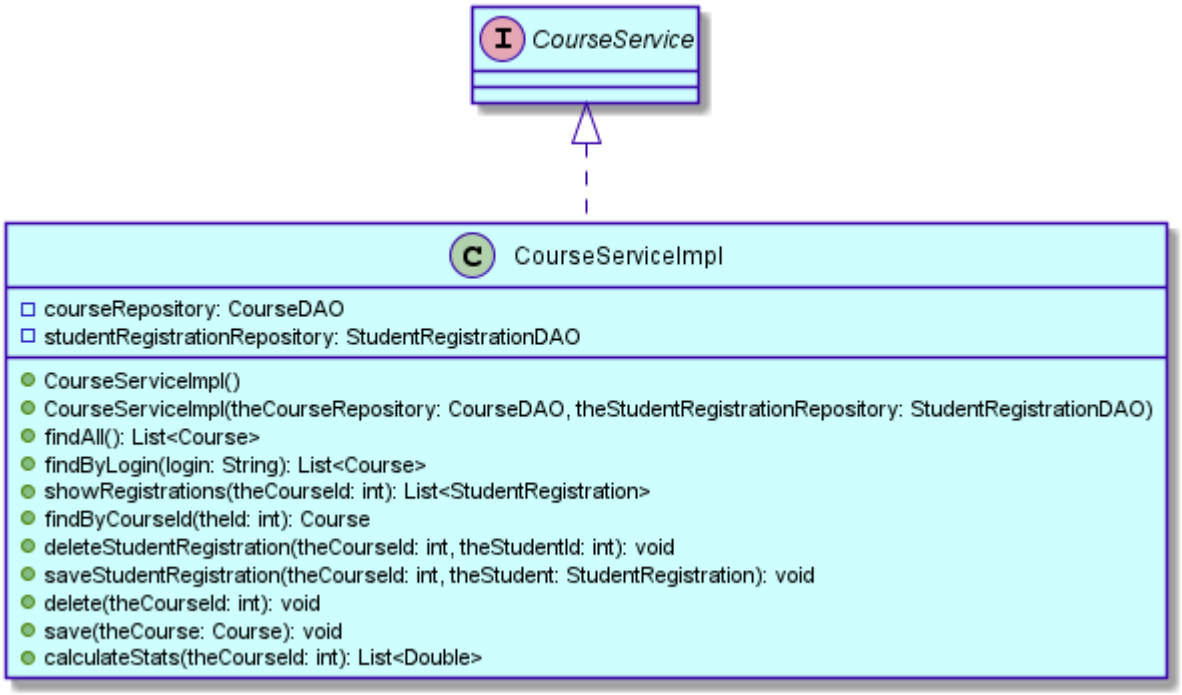
Class Name: StudentRegistration	
Responsibilities: ▪ Inherit the fields of the course template from the db ▪ Manage all the fields that are needed for every student registration(studentId,studentName,… etc)	**Collaborations:** ▪ Mysql db



Class Name: CourseService	
Responsibilities: ▪ Template for CourseServiceImpl	**Collaborations:** ▪ CourseServiceImpl



Class Name: CourseServiceImpl	
Responsibilities: ▪ Hand the list of courses to the html template that the user asked to see ▪ Find a course from the course repository(database) ▪ Delete one student registration for a certain course ▪ Save one student registration for a certain course ▪ If the user updates a student registration we save the new student registration ▪ Delete a specific course ▪ Save a specific course ▪ If the user updates a course we save the new course	**Collaborations:** ▪ Mysql db ▪ Course ▪ StudentRegistration



4.3 Use cases pairing to methods

UC1 : This use case is controlled by spring framework and is configured by our method ('configure') which belongs to the class DemoApplicationSecurityConfig which is placed in the com.example.demo.config package.

UC2 : This use case matches our method 'listCourses' which belongs to the class CourseController which is placed in the com.example.demo.controller package.

UC3 : This use case matches our method 'showFormForAdd' which belongs to the class CourseController which is placed in the com.example.demo.controller package.

UC4 : This use case matches our method 'delete' which belongs to the class CourseController which is placed in the com.example.demo.controller package.

UC5 : This use case matches our method 'showFormForUpdate' which belongs to the class CourseController which is placed in the com.example.demo.controller package.

UC6 : This use case matches our method 'showStudentRegistrations' which belongs to the class CourseController which is placed in the com.example.demo.controller package.

UC7 : This use case matches our method 'showFormForAddStudentRegistration' which belongs to the class CourseController which is placed in the com.example.demo.controller package.

UC8 : This use case matches our method 'deleteStudentRegistration' which belongs to the class CourseController which is placed in the com.example.demo.controller package.

UC9 : This use case matches our method 'showFormForUpdateStudentRegistration' which belongs to the class CourseController which is placed in the com.example.demo.controller package.

UC10 : This use case matches two of our methods. The first method is 'showFormForAddStudentRegistration' which belongs to the class CourseController which is placed in the com.example.demo.controller package, this happens because when we add a new student registration we want to register it's grades. The second method is 'showFormForUpdateStudentRegistration' which belongs to the class CourseController which is placed in the com.example.demo.controller package because when we update a student registration we can also update it's grades(re-register them).

UC11 : This use case matches our method 'calculateStatistics' which belongs to the class CourseController which is placed in the com.example.demo.controller package.

5 Testing

ID 1	testCourseControllerIsNotNull()	HappyDayScenario for CourseController.CourseController()
Description	ON	Any time
	RECEIVING	
	ENSURE	That the CourseController object has been successfully created
	OUTPUTS	A success message
	SUCH THAT	state is intact
Pre-cond.		The server is active
Input		
Output		
Post-cond.		
Method To test		CourseController.CourseController()

ID 2	testListCoursesReturnsPage()	HappyDayScenario for CourseController.listCourses()
Description	ON	A time when the user has log on
	RECEIVING	
	ENSURE	That the list-courses.html has been successfully displayed
	OUTPUTS	The list of courses
	SUCH THAT	state is intact
Pre-cond.		The user has log on
Input		
Output		The list of courses
Post-cond.		

Method To test		CourseController.listCourses()

ID 3	testSaveCoursesReturnPage()	HappyDayScenario for CourseController.saveCourse()
Description	ON	A time the list of courses is being displayed
	RECEIVING	The command to save a new course
	ENSURE	That the new course has been created and saved to the database
	OUTPUTS	A success message
	SUCH THAT	A new course has been added to the list of courses
Pre-cond.		The list of courses is being displayed
Input		Command to save a new course
Output		The new list of courses
Post-cond.		A new course has been created
Method To test		`courseController.saveCourse()`

ID 4	testListCoursesAddPage()	HappyDayScenario for CourseController.showFormForAdd()
Description	ON	A time the list of courses is being displayed
	RECEIVING	The command to save a new course
	ENSURE	That the form to add a new course is being displayed properly
	OUTPUTS	A success message
	SUCH THAT	state is intact
Pre-cond.		The list of courses is being displayed
Input		Command to save a new course
Output		The form to add a new course
Post-cond.		The user can see the form to add a new course
Method To test		`courseController.showFormForAdd()`

ID 5	testListCoursesUpdatePage()	HappyDayScenario for CourseController.showFormForUpdate()
Description	ON	A time when the list of courses is being displayed
	RECEIVING	The command to update an existing course
	ENSURE	That the update form displays the contents of the course properly
	OUTPUTS	A success message
	SUCH THAT	state is intact
Pre-cond.		The list of courses is being displayed and at least a course already exists
Input		The command to update a course

Output		The form to update a course
Post-cond.		The form to update a course is being displayed
Method To test		CourseController.showFormForUpdate()

ID 6	testCoursesUpdateValues()	HappyDayScenario for CourseController.showFormForUpdate() (and actually save it)
Description	ON	A time when the list of courses is being displayed
	RECEIVING	The command to save the updated form of a course
	ENSURE	That the course is saved to the database with it's updated form
	OUTPUTS	A success message
	SUCH THAT	A course has been updated
Pre-cond.		A course already exists in the database
Input		The command to update a course
Output		The form to update a course
Post-cond.		The course has been updated and the user now sees the list of courses updated
Method To test		CourseController.showFormForUpdate()

ID 7	testCoursesDelete()	HappyDayScenario for CourseController.delete()
Description	ON	A time when the user can see the list of courses and at least one course exists
	RECEIVING	The command to delete a course
	ENSURE	That the course has been successfully deleted from the database
	OUTPUTS	A success message
	SUCH THAT	A course has been successfully deleted
Pre-cond.		The user can see the list of courses and at least one course exists
Input		Command to delete a course
Output		
Post-cond.		The updated list of courses
Method To test		CourseController.delete()

ID 8	testListStudentRegistrations()	HappyDayScenario for CourseController.showStudentRegistrations()
Description	ON	A time when a course exists in the list of courses
	RECEIVING	The user requests to see it's student registrations
	ENSURE	That the list of student registrations is being properly displayed
	OUTPUTS	A success message
	SUCH THAT	The list of student registrations of a course is being

		displayed
Pre-cond.		A course exists and the user is seeing the list of courses
Input		Command to see list of student registrations of a course
Output		The list of student registrations
Post-cond.		The user is seeing the list of student registrations
Method To test		CourseController.showStudentRegistrations()

ID 9	testUpdateStudentRegistrations()	HappyDayScenario for CourseController.showFormForUpdateStudentRegistration()
Description	ON	A time when the user is seeing the list of student registrations
	RECEIVING	The command to update a student registration
	ENSURE	That the student registration is successfully updated
	OUTPUTS	A success message
	SUCH THAT	A student registration has been updated
Pre-cond.		The user is seeing the list of student registrations and a student registrations already exists in a course
Input		Command to update a student registration
Output		Success message
Post-cond.		The student registration has been updated and the user now sees the updated list of student registrations
Method To test		CourseController.showFormForUpdateStudentRegistration()

ID 10	testStudentRegistrationDelete()	HappyDayScenario for CourseController.deleteStudentRegistration()
Description	ON	A time when the user is seeing the list of student registrations of a course
	RECEIVING	The command to delete a student registration
	ENSURE	That the student registration has been deleted
	OUTPUTS	A success message
	SUCH THAT	The student registration has been deleted
Pre-cond.		The user is seeing the list of student registrations and at least one exists
Input		Command to delete a student registration
Output		Success message
Post-cond.		The updated list of student registrations
Method To test		CourseController.deleteStudentRegistration()

ID 11	testCalculateStatistics()	HappyDayScenario for CourseController.calculateStatistics()
Description	ON	A time when the user is seeing the list of courses

	RECEIVING	The command to calculate the statistics of a specific course
	ENSURE	That the reports file has been successfully created
	OUTPUTS	The statistics requested from the user
	SUCH THAT	The statistics are properly calculated and displayed
Pre-cond.		There is at least one course with at least one student registration inside the database and the user is seeing the list of courses.
Input		Command to calculate the course statistics
Output		The statistics requested
Post-cond.		The statistics have been displayed and the user can go back to the list of courses
Method To test		CourseController.calculateStatistics()

ID 12	testSaveCourseToDaoFindById()	HappyDayScenario for CourseDao.findById()
Description	ON	Any time
	RECEIVING	The command from the system to find a course in the database
	ENSURE	That the course is found
	OUTPUTS	
	SUCH THAT	The course is found
Pre-cond.		
Input		Command to find a course from the database
Output		
Post-cond.		The course has been found
Method To test		`CourseDao.findById()`

6 Notices

During our time making this software, our goal was to implement all the user stories that was requested. Due to lack of time, we didn't manage to put some add-ons that would make it fully functional (based on what was requested). For example:

a) we could have make our software catch and correct bad input from the user such as:

1) in a field that required an Integer we assume that the user inputs only an Integer

2) in the fields that require the weights for the project grade and the exams grade of a course, we assume that the user will enter the number in the correct format in the range [0,1] and that they will add up to 1

b) we could also have made our software handle the users properly so that:

1) the admin will have full privileges over all of the courses and student registrations

2) the admin will have full privileges over the accounts of the professors

3) the professors will only be able to manage the courses they teach