CoursesManagementApp

Sprint Report

Christos Karagiannidis 4375
Nefeli-Eleni Katsilerou 4385
Fotini Ifanti 4516

VERSIONS HISTORY

Date	Version	Description	Author
26/2/2022 - 27/2/2022	V0.0	Reading and comprehending project requirements	CK , NK , FY
2/3/2022 - 6/3/2022	V1.0	Database creation	CK , NK , FY
8/3/2022 – 18/3/2022	V2.0	Creation of entities, controllers ,services and DAOs	CK , NK , FY
23/3/2022 – 1/4/2022	V2.1	Connection of Java with our DB	CK , NK , FY
6/4/2022 – 13/4/2022	V2.2	Implementation of user stories	CK , NK , FY
23/4/2022 – 2/5/2022	V3.0	HTML implementation	CK , NK , FY
3/5/2022 – 10/5/2022	V4.0	JUnit Testing	CK , NK , FY
12/5/2022 – 15/5/2022	V4.1	Sprint report creation and video recording	CK , NK , FY

1 Introduction

1.1 Purpose

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

1.2 Document Structure

The rest of this document is structured as follows. Section 2 describes our Scrum team and specifies the this Sprint's backlog. In Section 3 we list the sprints that we performed and the user stories that have been realized in each Sprint. Section 4 includes the project design providing UML package and class diagrams.

2 Scrum team and Sprint Backlog

2.1 Scrum team

Product Owner	Christos Karagiannidis
Scrum Master	Nefeli-Eleni Katsilerou , Fotini Ifanti
Development Team	Christos Karagiannidis, Nefeli-Eleni Katsilerou , Fotini Ifanti

2.2 Sprints

Sprint No	Begin Date	End Date	Number of weeks	User stories
1	**6/4/2022**	**9/4/2022**	**0**	**US2,US3,US4,US5**
2	**9/4/2022**	**10/4/2022**	**0**	**US6,US7,US8,US9**

3	**11/4/2022**	**11/4/2022**	**0**	**US10,US11**
4	**12/4/2022**	**12/4/2022**	**0**	**US12**
5	**13/4/2022**	**13/4/2022**	**0**	**US1**

3 Use Cases

3.1 Login

Use case ID	1
Actors	Instructor
Pre conditions	Entered correct username and password in the corresponding fields
Main flow of events	1. The user story begins when the instructor enters his login credentials 2. Thymeleaf spring security verifies that the credentials given correspond to a USER with a role ADMIN in the DB.
Alternative flow 1	If credentials do not match any user in DB then the login page is reloaded and a ‘wrong username or password’ message is reported.
Alternative flow 2	
Post conditions	Instructor is redirected to the home page (course list).

3.2 List all courses

Use case ID	2
Actors	Instructor
Pre conditions	A few courses exist in the DB
Main flow of events	1. User story begins when the Instructor logs in. 2. Java calls the course service function findAll() and saves the courses

	returned to a list of Courses. 3. An attribute is added to the model containing the courses saved in the previous step
Alternative flow 1	DB doesn't contain any courses so a blank list is returned
Alternative flow 2	…….
Post conditions	The corresponding HTML file is returned returning the results to the Instructor

3.3 Delete a course

Use case ID	3
Actors	Instructor
Pre conditions	A course exists in order to be deleted.
Main flow of events	1. User story begins when the Instructor presses the red delete button under the 'Actions' tab. 2. Java creates a list with every student Registration associated to this course. 3. Those student registrations are deleted. 4. The course is deleted
Alternative flow 1	
Alternative flow 2	…….
Post conditions	Course list is reloaded with the new courses (without the deleted one).

3.4 Add a course

Use case ID	4
Actors	Instructor

Pre conditions	
Main flow of events	1. User story begins when the Instructor presses the 'Add course' button. 2. A new course item is created in Java. 3. Course-form HTML file is returned 4. Instructor fills the blank fields and presses 'Save' to submit 5. Function/Method saveCourse is called
Alternative flow 1	
Alternative flow 2	....
Post conditions	Course is saved in the DB and course list page is reloaded.

3.5 Update a course

Use case ID	5
Actors	Instructor
Pre conditions	A course exists in the DB in order to be Updated
Main flow of events	1. User story begins when the Instructor presses the 'update' button in the actions tab. 2. Java calls the course service function findbyId() and saves the course returned. 3. That course is added to the model. 4. Course-form HTML file is returned 5. Instructor fills the blank fields and presses 'Save' to submit.
Alternative flow 1	Wrong input type given in fields (ex. String where float is expected) produces error.
Alternative flow 2	....
Post	The updated course list is returned

conditions	

3.6 Add a student registration

Use case ID	6
Actors	Instructor
Pre conditions	A course exists in order to assign a proper value to the course id field.
Main flow of events	1. User story begins when the Instructor presses the 'Add student registration' button 2. Java creates a new StudentRegistration item and automatically sets its course id field to the id of the course next to which the instructor pressed the 'Add student registration' button in step 1. 3. That item is added to the model 4. The student registration form HTML file is returned 5. The instructor fills in the rest blank fields (course id field is already assigned) and presses 'Save' 6. Function/Method saveStudentRegistration() is called
Alternative flow 1	DB doesn't contain any courses so a blank list is returned
Alternative flow 2	....
Post conditions	The student registration is saved in the DB and then the list of all the student registrations is reloaded.

3.7 Delete a course

Use case ID	7
Actors	Instructor
Pre conditions	A student registration exists in order to be deleted.
Main flow of	1. User story begins when the Instructor presses the red delete

events	button under the ‘Actions’ tab. 2. The student registration is deleted
Alternative flow 1	
Alternative flow 2	…….
Post conditions	Instructor is redirected to the list that contains all student registrations (except the deleted one)

3.8 Update a student registration

Use case ID	8
Actors	Instructor
Pre conditions	A student registration exists in the DB (associated with an already existing course) in order to be Updated
Main flow of events	1. User story begins when the Instructor presses the ‘update’ button in the actions tab. 2. Java calls the student registration service function findbyId() and saves the student registration returned. 3. That student registration is added to the model. 4. Student registration-form HTML file is returned 5. Instructor fills the blank fields and presses ‘Save’ to submit.
Alternative flow 1	Wrong input type given in fields (ex. String where float is expected) produces error.
Alternative flow 2	…….
Post conditions	The updated student registration list is returned

3.9 Calculate overall/final grades

Use case ID	9
Actors	Instructor

Pre conditions	There is at least one student to calculate overall grades for.
Main flow of events	1. User story begins when the Instructor presses the ‘calculate grades’ button in the actions tab. 2. Java calls the student registration service function findAllByCourseid() and saves the student registrations returned in a list. 3. For every student registration in the list a final grade is calculated and added to its finalgrade field. 4. All student registrations in the list are added to the model.
Alternative flow 1	
Alternative flow 2	....
Post conditions	A list of all student registrations in this particular course is returned containing project ,exam and final grades for each student.

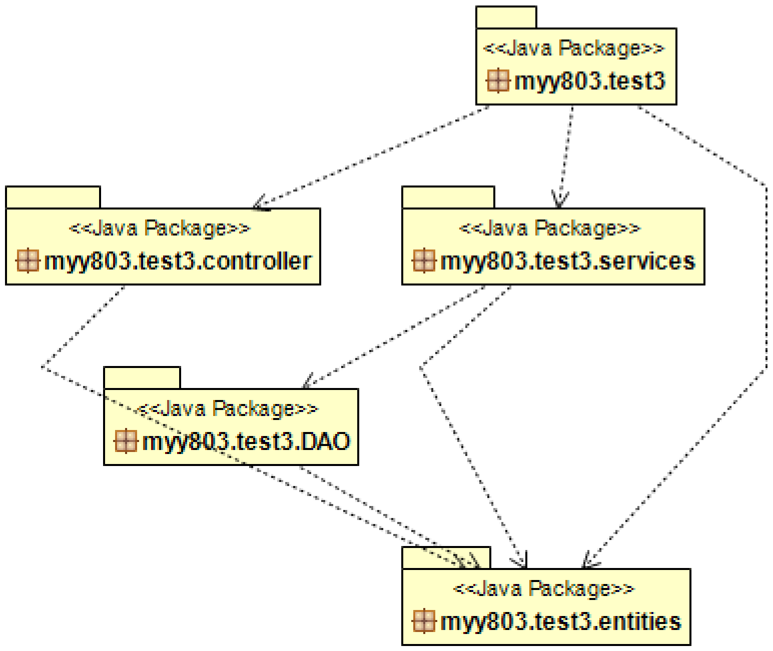
3.10 Calculate statistics

Use case ID	10
Actors	Instructor
Pre conditions	
Main flow of events	1. User story begins when the Instructor presses the ‘calculate statistics’ button . 2. Java calls the student registration service function findAll() and saves the student registrations returned. 3. Min, Max, mean, stdDev, variance, percentiles, skewness, kurtosis and median are calculated for each grade type. 4. Exam stats, Project stats and final stats are added to the model
Alternative flow 1	Not enough student grades exist so some fields return 0 value or NaN
Alternative flow 2	....

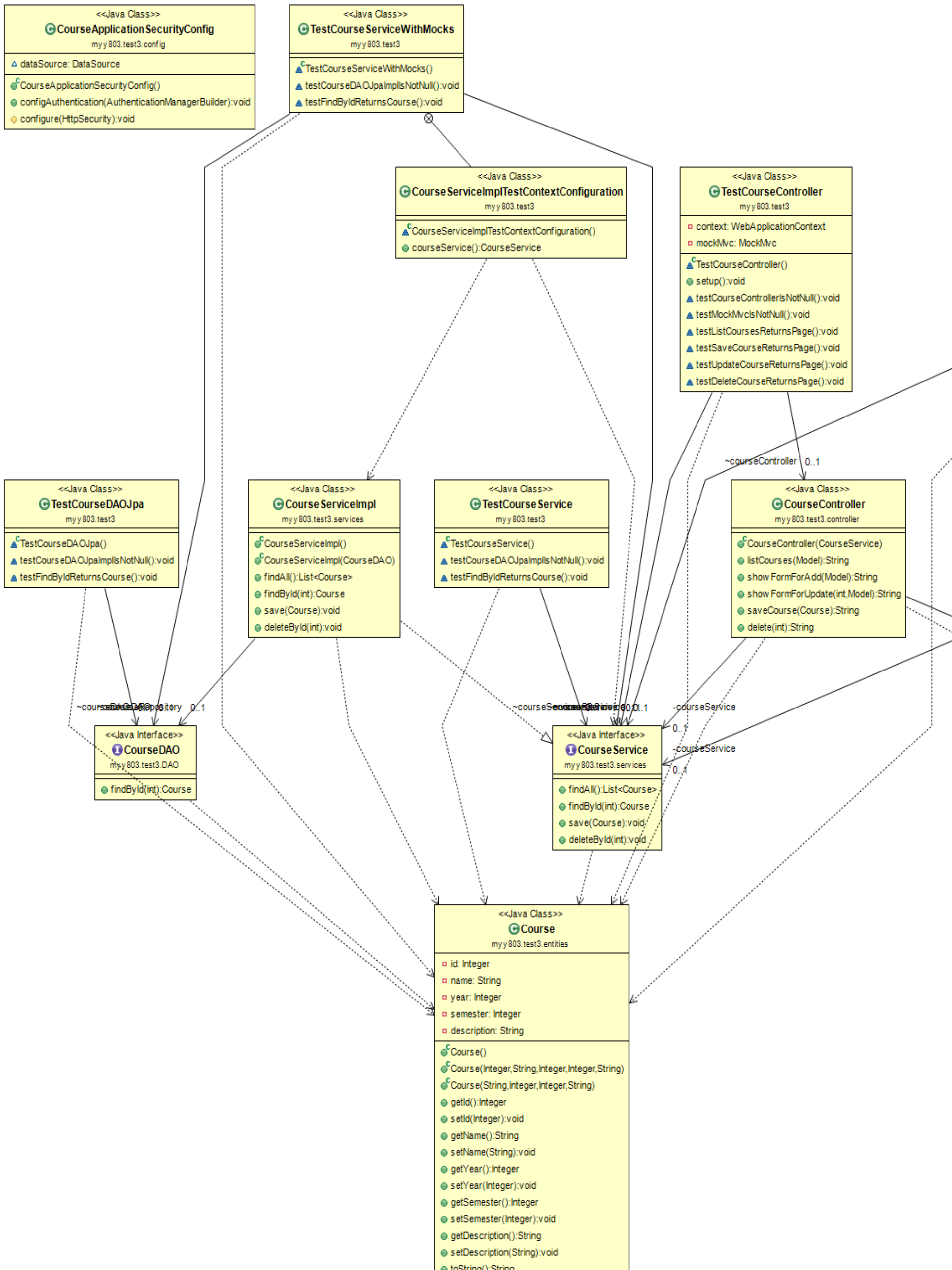
Post conditions	Stats for the three grade types are shown on screen.

4 Design

4.1 Architecture



4.2 Design



<<Java Class>>
CourseApplicationSecurityConfig
myy803.test3.config
dataSource: DataSource
CourseApplicationSecurityConfig()
configAuthentication(AuthenticationManagerBuilder):void
configure(HttpSecurity):void
<<Java Class>>
TestCourseServiceWithMocks
myy803.test3
TestCourseServiceWithMocks()
testCourseDAOJpaImplIsNotNull():void
testFindByIdReturnsCourse():void
<<Java Class>>
CourseServiceImplTestContextConfiguration
myy803.test3
CourseServiceImplTestContextConfiguration()
courseService():CourseService
<<Java Class>>
TestCourseController
myy803.test3
context: WebApplicationContext
mockMvc: MockMvc
TestCourseController()
setup():void
testCourseControllerIsNotNull():void
testMockMvcIsNotNull():void
testListCoursesReturnsPage():void
testSaveCourseReturnsPage():void
testUpdateCourseReturnsPage():void
testDeleteCourseReturnsPage():void
~courseController
0..1
<<Java Class>>
TestCourseDAOJpa
myy803.test3
TestCourseDAOJpa()
testCourseDAOJpaImplIsNotNull():void
testFindByIdReturnsCourse():void
<<Java Class>>
CourseServiceImpl
myy803.test3.services
CourseServiceImpl()
CourseServiceImpl(CourseDAO)
findAll():List<Course>
findById(int):Course
save(Course):void
deleteById(int):void
<<Java Class>>
TestCourseService
myy803.test3
TestCourseService()
testCourseDAOJpaImplIsNotNull():void
testFindByIdReturnsCourse():void
<<Java Class>>
CourseController
myy803.test3.controller
CourseController(CourseService)
listCourses(Model):String
showFormForAdd(Model):String
showFormForUpdate(int,Model):String
saveCourse(Course):String
delete(int):String
0..1
0..1
-courseService
0..1
-courseService
0..1
<<Java Interface>>
CourseDAO
myy803.test3.DAO
findById(int):Course
<<Java Interface>>
CourseService
myy803.test3.services
findAll():List<Course>
findById(int):Course
save(Course):void
deleteById(int):void
<<Java Class>>
Course
myy803.test3.entities
id: Integer
name: String
year: Integer
semester: Integer
description: String
Course()
Course(Integer,String,Integer,Integer,String)
Course(String,Integer,Integer,String)
getId():Integer
setId(Integer):void
getName():String
setName(String):void
getYear():Integer
setYear(Integer):void
getSemester():Integer
setSemester(Integer):void
getDescription():String
setDescription(String):void
toString():String

<Document the classes that are included in this release in terms of CRC cards according to the template that is given below.>

Class Name: Course	
Responsibilities: ▪ Construct a course entity. ▪ Associate course's java fields to mySQL columns. ▪ Create setters and getters.	**Collaborations:**

Class Name: StudentRegistration	
Responsibilities: ▪ Construct a studentRegistration entity ▪ Associate studentRegistration's fields to mySQL columns. ▪ Create setters and getters	**Collaborations:**

Class Name: CourseController	
Responsibilities: ▪ Map URLs to functions/methods ▪ List all existing courses ▪ Add a new course ▪ Update an already existing course ▪ Delete an already existing course (first by deleting all existing studentRegistrations associated with that course) ▪ Save a course by calling course service function 'save'. ▪ Delete a course by calling course service function 'deletebyid'. ▪ Return HTML files for each function/method	**Collaborations:** ▪ Course service ▪ Student Registration service

Class Name: StudentRegistration Controller	
Responsibilities: ▪ Map URLs to functions/methods ▪ List all student Registrations ▪ List all student registrations associated with a particular course ▪ Add a new student registration ▪ Update an already existing student registration ▪ Delete an already existing student registration ▪ Calculate Final grade with respect to a weighted average ▪ Show detailed statistics for a particular course's exam grades, project grades and final grades. ▪ Save a student registration by calling studentRegistration service function 'save' ▪ Delete a student registration by calling student registration service function 'deletebyid' ▪ Return HTML files for each function/method	**Collaborations:** ▪ Course service ▪ Student Registration service

Class Name: StudentRegistration service impl	
Responsibilities: ▪ Implement studentRegistration service's methods ▪ Create a student registration repository ▪ Find all student registrations ▪ Find a particular student registration by id ▪ Find all student registrations associated with a particular course id ▪ Save a student registration to the	**Collaborations:** ▪ studentRegistrationDAO ▪ studentRegistration service

student registration repository ▪ Delete a student registration from the student registration repository ▪ Delegates the appropriate calls to the studentRegistration DAO	

Class Name: Course service impl	
Responsibilities: ▪ Implement course service's methods ▪ Create a course repository ▪ Find all courses ▪ Find a particular course by id ▪ Save a course to the course repository ▪ Delete a course from the course repository ▪ Delegates the appropriate calls to the course DAO	**Collaborations:** ▪ courseDAO ▪ course service