CoursesManagementApp

Sprint Report

Panagiotis Kolokouris 4914

VERSIONS HISTORY

Date	Version	Description	Author
28/03	1	Kick-off document	PK
08/05	1.1	First draft	PK
17/05	1.2	Final draft	PK

1 Introduction

This document provides information concerning the development of the project called “Course Management Application”.

1.1 Purpose

The purpose of this project is to develop a Web application that allows an instructor to manage grading of the courses that they teach.

An instructor will be able to access the application via their log-in information. The application allows many instructor registrations via a register page.

Upon log-in the instructor will be presented with their recorded courses and their accompanying attributes. This information will be stored in a MySQL database. The instructor will be able to add, edit and remove courses at will.

Furthermore, the instructor will be able to manage, for each course recorded, the student registrations associated with the course. Student registrations are also stored in the same database along with their attributes. The instructor will be able to add, edit and remove student registrations for each course.

Finally, the application accommodates the calculation of the students’ final grades, based on the project-exam weights, and the calculation of descriptive statistics based on the final grades for each course.

1.2 Document Structure

The rest of this document is structured as follows.

- Section 2 describes the Scrum team and specifies this Sprint's backlog.
- Section 3 describes the application’s use cases.
- Section 4 specifies the main design concepts for this release of the project.
- Section 5 describes the tests developed for each class of each layer.
- Section 6 lists out all the assumptions that were made during the development process along with the limitations (to do) items for future development.

2 Scrum team and Sprint Backlog

2.1 Scrum team

The scrum team which will undertake this project is presented in Table 2.1.

Table 2.1: Scrum team

Product Owner	Apostolos Zarras
Scrum Master	Panagiotis Kolokouris
Development Team	Panagiotis Kolokouris

2.2 Sprints

The software development phase has been split into three sprints, each of which lasted two weeks. The sprint schedule is presented in Table 2.2.

Table 2.2: Sprint Schedule

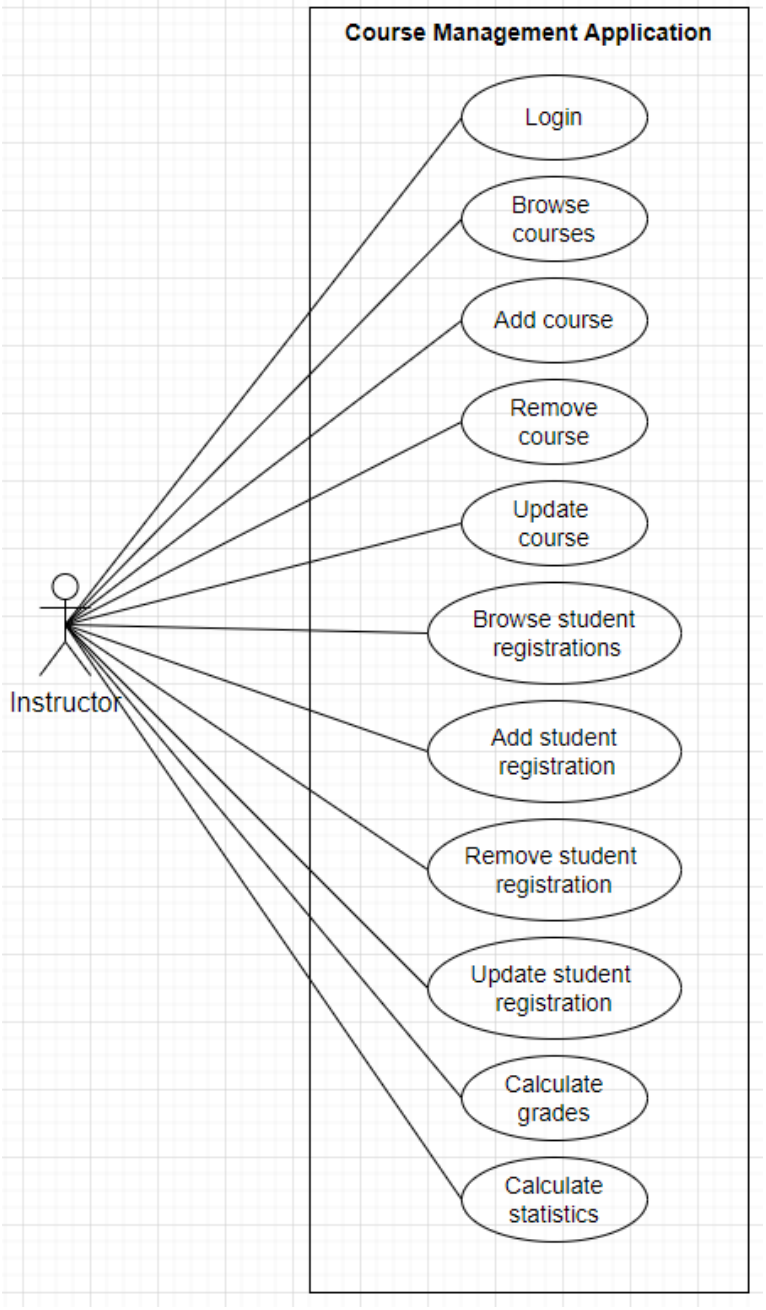
Sprint No	Begin Date	End Date	Number of weeks	User stories
1	**28/03**	**10/04**	**2**	**US1, US2, US6**
2	**11/04**	**24/04**	**2**	**US3, US4, US5, US7, US8, US9, US10**
3	**25/04**	**08/05**	**2**	**US11-US12**

3 Use Cases

The requirements of the application derive from the provided abstract user stories. In this section the user stories are formulated in more detailed use cases.

The UML use case diagram is presented in Figure 3.1. The detailed use cases are described in the following sections.

Figure 3.1:UML Use Case Diagram



3.1 UC1 - Login

Use case ID	UC1
Actors	Instructor
Preconditions	None
Main flow of events	1. The use case starts when the user launches the application. 2. The user enters username. 3. The user enters password. 4. The user presses “Login” 5. If username and password are correct, the system logs the user in.
Alternative flow 1	If the user presses “Login” without entering a username, the user is prompted to re-try.
Alternative flow 2	If username does not exist in database, the user is prompted to re-try.
Alternative flow 3	If the password the user entered does not match the password stored in the database, the user is prompted to re-try.
Post conditions	None

3.2 UC2 – Browse Courses

Use case ID	UC2
Actors	Instructor
Preconditions	The user has successfully logged in.
Main flow of events	1. The use case starts when the user has successfully logged in. 2. The system displays the courses associated with the logged in user.
Alternative flow 1	None
Alternative flow 2	None
Post conditions	None

3.3 UC3 – Add Course

Use case ID	UC3
Actors	Instructor
Preconditions	The user has successfully logged in.
Main flow of events	1. The use case starts when the user presses the “Add New” button on the courses’ page. 2. The system displays a form with the required attributes of a course. 3. The user fills in all course attributes. 4. The user presses “Add Course”. 5. The system adds the course to the list of courses. 6. The system shows the courses’ page.
Alternative flow 1	If the user fails to fill in all fields, the user is prompted to re-try.
Alternative flow 2	If the user enters an invalid entry to any field, the user is prompted to re-try.
Alternative flow 3	If the user presses “Cancel”, the system shows the courses’ page.
Post conditions	The system updates the database and the course’ page to include the new course.

3.4 UC4 – Remove Course

Use case ID	UC4
Actors	Instructor
Preconditions	The user has successfully logged in.
Main flow of events	1. The use case starts when the user presses the “Del” button next to a course. 2. The system removes the course from the list of courses. 3. The system shows the courses page.
Alternative flow 1	None
Alternative flow 2	None
Post conditions	The system updates the database and the course’ page to erase the deleted course.

3.5 UC5 – Update Course

Use case ID	UC5
Actors	Instructor
Preconditions	The user has successfully logged in.
Main flow of events	1. The use case starts when the user presses the "Edit" button next to a course. 2. The system displays a form with the current attributes of a course. 3. The user amends zero or more of the course attributes. 4. The user presses "Save Course". 5. The system updates the course in the list of courses. 6. The system shows the courses page.
Alternative flow 1	If the user leaves a field empty, the user is prompted to re-try.
Alternative flow 2	If the user enters an invalid entry to any field, the user is prompted to re-try.
Alternative flow 3	If the user presses "Cancel", the system shows the courses' page.
Post conditions	The system updates the database and the course' page with the updated course.

3.6 UC6 – Browse Students

Use case ID	UC6
Actors	Instructor
Preconditions	The user has successfully logged in.
Main flow of events	1. The use case starts when the user presses the "Show" button next to a course. 2. The system displays the student registrations page with all the student registrations associated with the course the user chose.
Alternative flow 1	None
Alternative flow 2	None
Post conditions	None

3.7 UC7 – Add Student

Use case ID	UC7
Actors	Instructor
Preconditions	The user has pressed the “Show” button next to a course.
Main flow of events	1. The use case starts when the user presses the “Add New” button on the student registrations’ page. 2. The system displays a form with the required attributes of a student registration. 3. The user fills in all student registration attributes. 4. The user presses “Add Student”. 5. The system adds the student registration to the list. 6. The system shows the student registrations’ page.
Alternative flow 1	If the user fails to fill in all fields, the user is prompted to re-try.
Alternative flow 2	If the user enters an invalid entry to any field, the user is prompted to re-try.
Alternative flow 3	If the user presses “Cancel”, the system shows the student registrations’ page.
Post conditions	The system updates the database and the student registrations’ page to include the new student registration.

3.8 UC8 – Remove Student Registration

Use case ID	UC8
Actors	Instructor
Preconditions	The user has pressed the “Show” button next to a course.
Main flow of events	1. The use case starts when the user presses the “Del” button next to a student registration. 2. The system removes the student registration from the list. 3. The system shows the student registrations’ page.
Alternative flow 1	None
Alternative flow 2	None
Post conditions	The system updates the database and the student registration’ page to erase the deleted student registration.

3.9 UC9 – Update Student Registration

Use case ID	UC9
Actors	Instructor
Preconditions	The user has pressed the “Show” button next to a course.
Main flow of events	1. The use case starts when the user presses the “Edit” button next to a student registration. 2. The system displays a form with the current attributes of a student registration. 3. The user amends zero or more of the student registration attributes. 4. The user presses “Save Student Registration”. 5. The system updates the student registration in the list. 6. The system shows the student registrations’ page.
Alternative flow 1	If the user leaves a field empty, the user is prompted to re-try.
Alternative flow 2	If the user enters an invalid entry to any field, the user is prompted to re-try.
Alternative flow 3	If the user presses “Cancel”, the system shows the student registration’ page.
Post conditions	The system updates the database and the student registrations’ page with the updated student registration.

3.10 UC10 – Calculate Grades

Use case ID	UC10
Actors	Instructor
Preconditions	The user has pressed the “Show” button next to a course.
Main flow of events	1. The use case starts when the user presses the “Calculate Final Grades” button on the student registration’s page. 2. The system calculates the students’ final grades based on the exam – project weights the user has provided. 3. The system shows the student registrations’ page where column “Grade” now shows the students’ final grades.
Alternative flow 1	None
Alternative flow 2	None
Post conditions	The system updates the database and the student registrations’ page with the calculated grades.

3.11 UC11 – Calculate Grades

Use case ID	UC11
Actors	Instructor
Preconditions	The user has pressed the “Show” button next to a course.
Main flow of events	1. The use case starts when the user presses the “Calculate Grade Statistics” button on the student registration’s page. 2. The system calculates descriptive statistics for the students’ final grades. 3. The system shows the statistics’ page.
Alternative flow 1	None
Alternative flow 2	None
Post conditions	None

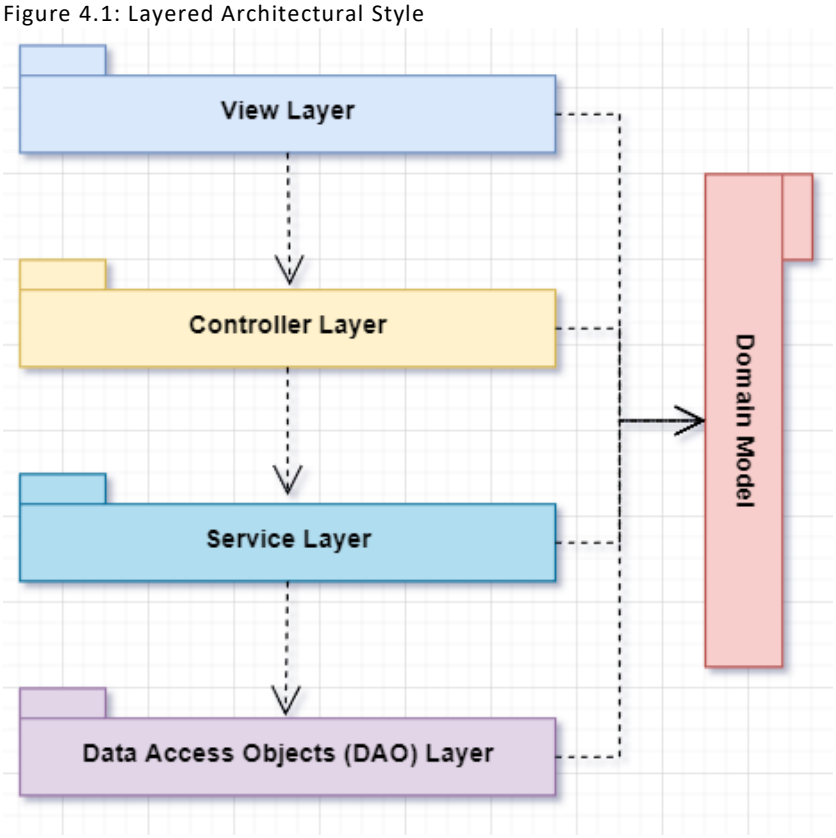
4 Design

This section presents the detailed design of the application.

4.1 Architecture

The software's architecture conforms with the layered architectural style presented in Figure 4.1.

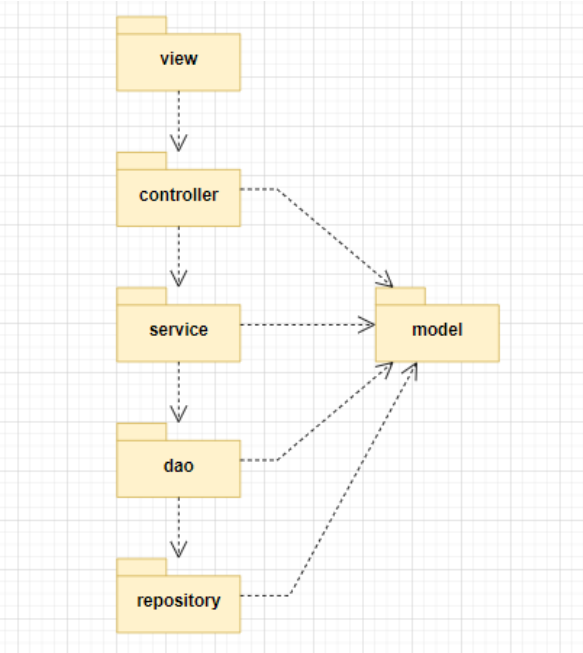
Figure 4.1: Layered Architectural Style



The software architecture in terms of UML package diagram is presented in Figure 4.2.

It must be noted here that the view package does not actually hold Java classes. It consists of a series static and dynamic resources such as html templates, css styles and image files.

Figure 4.2: UML Package Diagram



4.2 Design

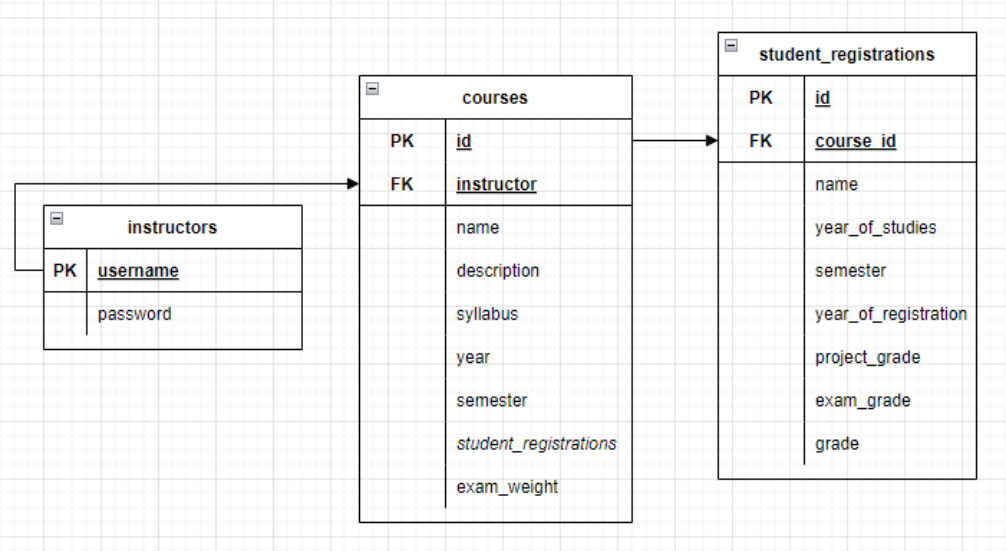
The application's design discussion is divided into two parts, the following:

- Conceptual modelling i.e., database schema diagram and UML class diagrams which are presented in section 4.2.1.
- CRC cards for Java classes which are presented in section 4.2.2.

4.2.1 CONCEPTUAL MODELLING

The application requires the development of a relational database for the storing, updating, and retrieving of information. This relational database was developed in MySQL. The database schema is presented in Figure 4.3.

Figure 4.3: Relational Database Schema

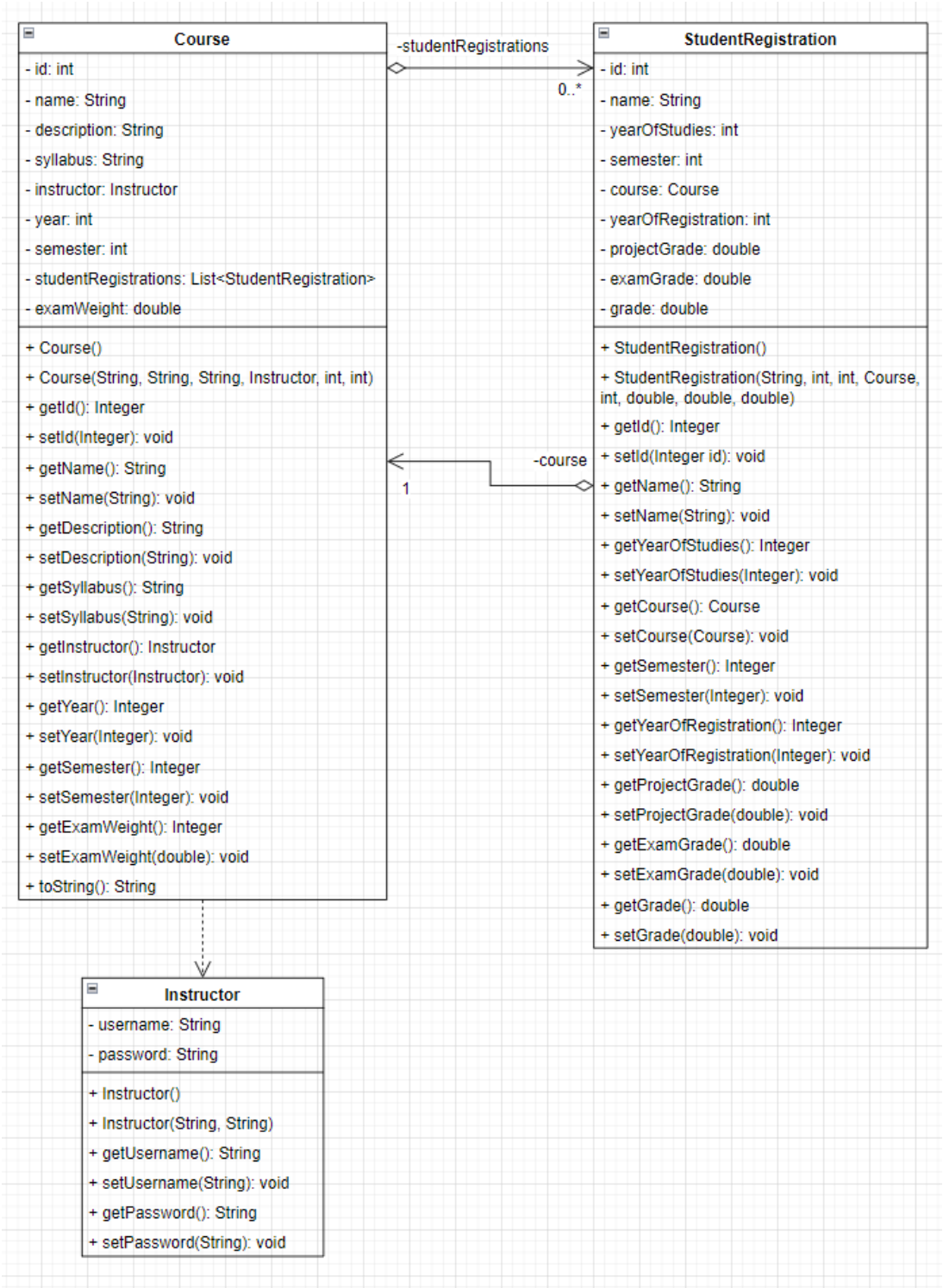


The following sections present the detailed application design in terms of UML class diagrams for each package containing Java classes.

4.2.1.1 MODEL

The UML class diagram for the model package is seen in Figure 4.4.

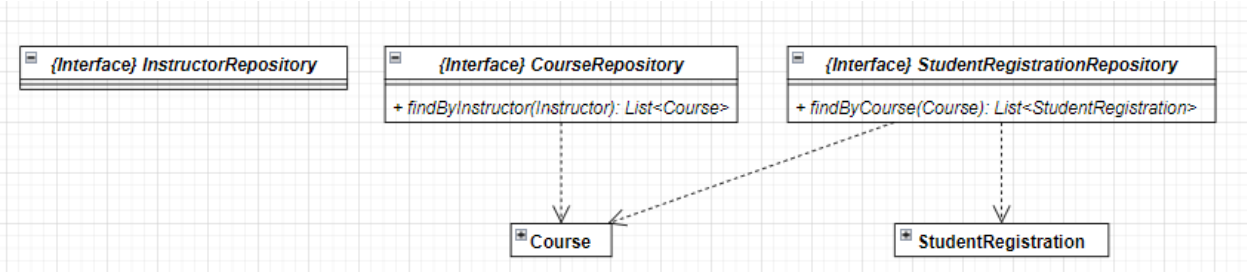
Figure 4.4: Model class diagram



4.2.1.2 REPOSITORY

The UML class diagram for the repository package is seen in Figure 4.5. The classes external to the package i.e., Instructor, Course and StudentRegistration are presented in an abstract view as they have been fully described in Figure 4.4.

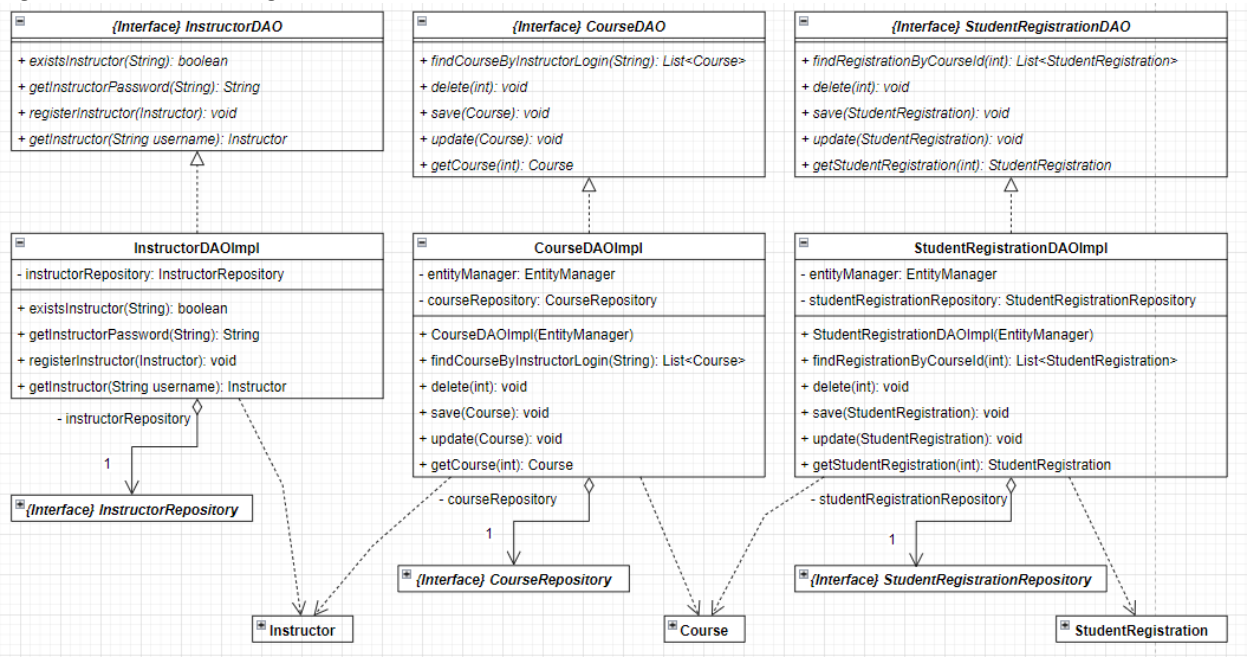
Figure 4.5:Repository class diagram



4.2.1.3 DATA ACCESS OBJECTS (DAO)

The UML class diagram for the dao package is seen in Figure 4.6. The classes external to the package are presented in an abstract view as they have been fully described in previous figures.

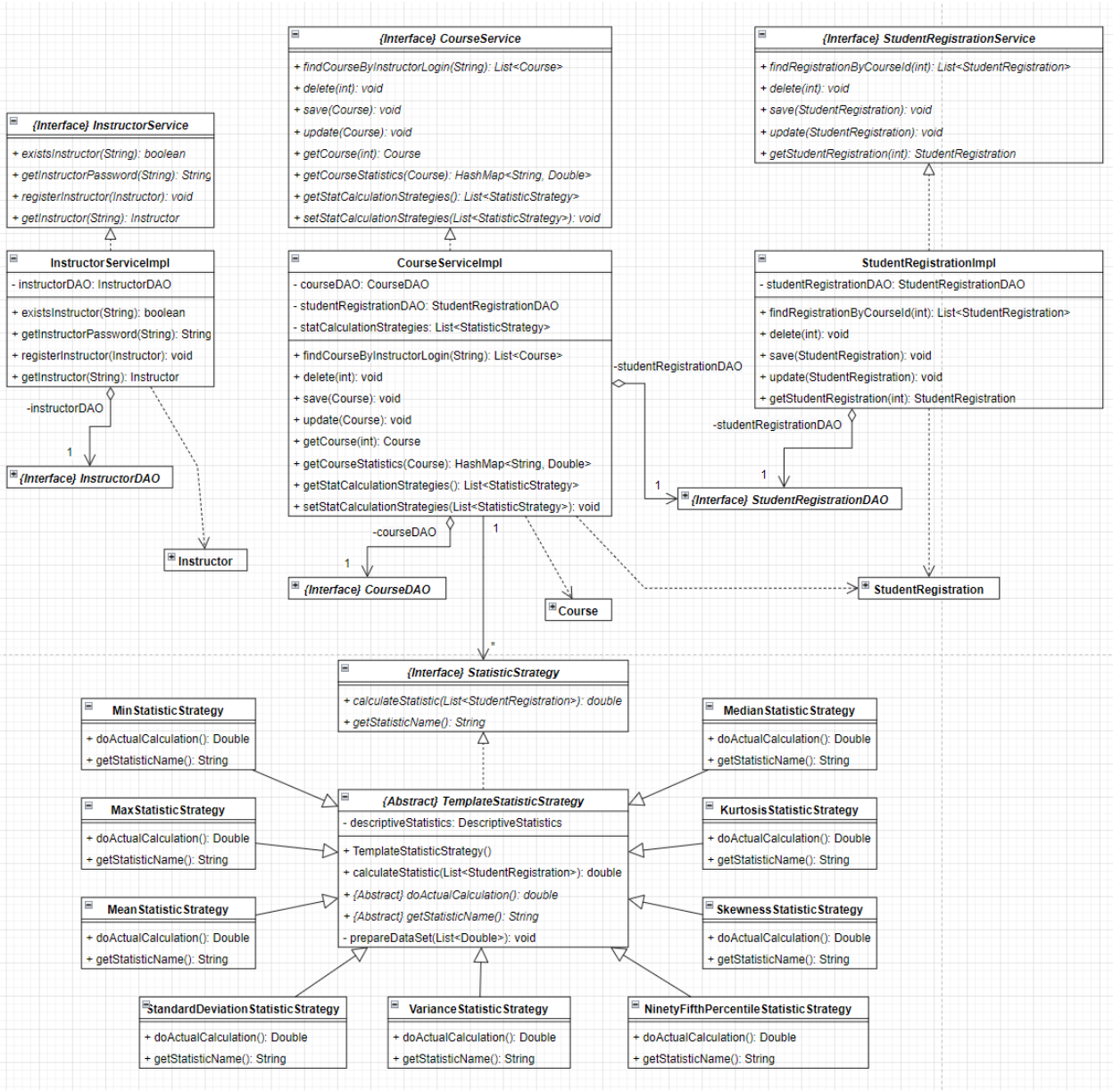
Figure 4.6:DAO class diagram



4.2.1.4 SERVICE

The UML class diagram for the service package is seen in Figure 4.7. The classes external to the package are presented in an abstract view as they have been fully described in previous figures.

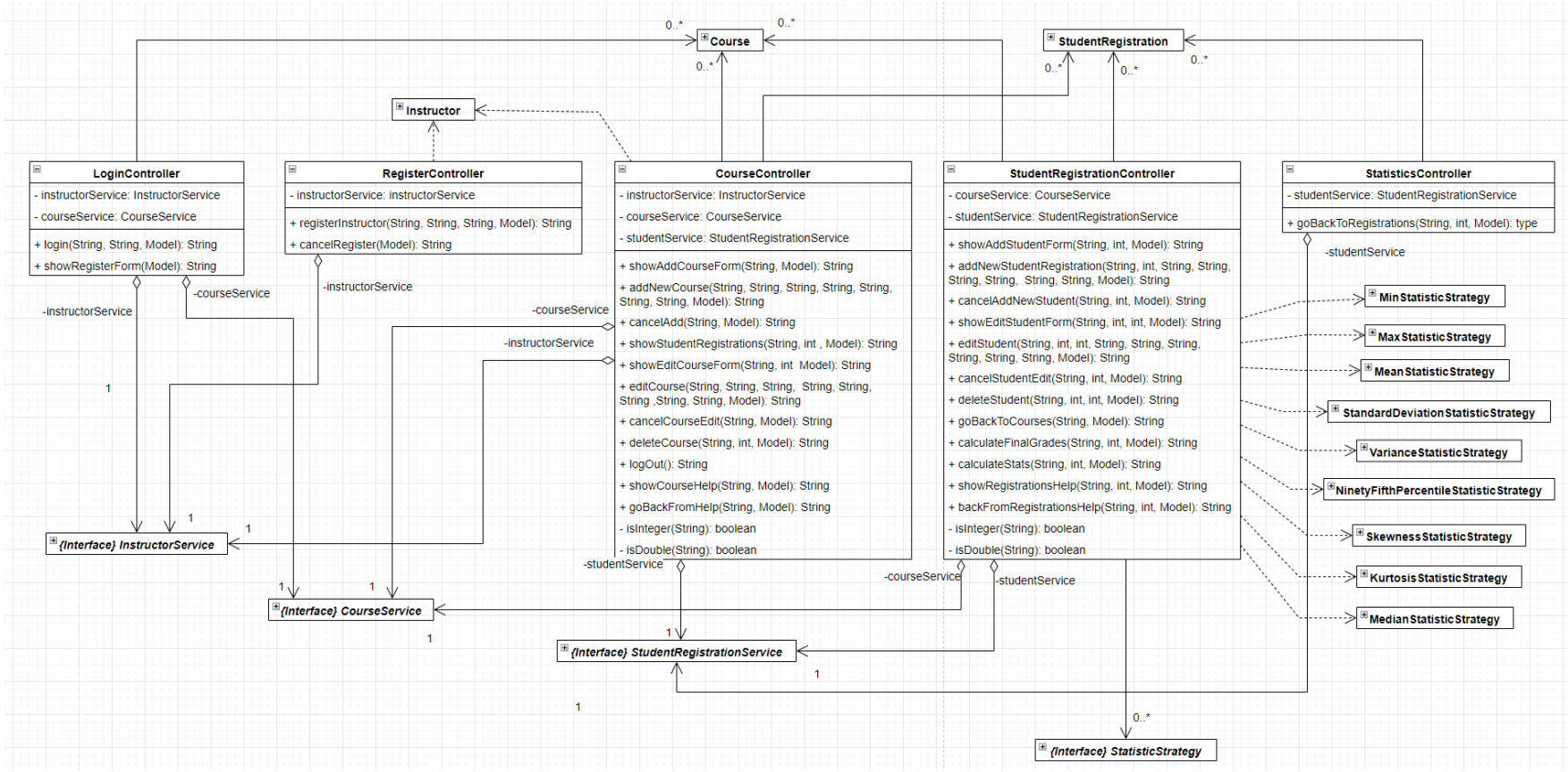
Figure 4.7: Service class diagram



4.2.1.5 CONTROLLER

The UML class diagram for the controller package is seen in Figure 4.8. The classes external to the package are presented in an abstract view as they have been fully described in previous figures.

Figure 4.8: Controller class diagram



4.2.2 CRC CARDS

The application's Java classes are discussed in more detail in the following sections in terms of CRC cards.

4.2.2.1 MODEL

The model package classes are documented in tables Table 4.1 to Table 4.3.

Table 4.1: CRC Card for the Instructor class

Class Name: Instructor	
Responsibilities: ▪ Holds the instructor information i.e., username and password in private fields. ▪ Provides set and get methods for each field for the Spring Java Persistent Access (JPA).	**Collaborations:** ▪ none

Table 4.2: CRC Card for the Course class

Class Name: Course	
Responsibilities: ▪ Holds the course information i.e., id, name, description etc. in private fields. ▪ Provides set and get methods for each field for the Spring Java Persistent Access (JPA).	**Collaborations:** ▪ StudentRegistration

Table 4.3: CRC Card for the StudentRegistration class

Class Name: StudentRegistration	
Responsibilities: ▪ Holds the student registration information i.e., id, name, semester etc. in private fields. ▪ Provides set and get methods for each field for the Spring Java Persistent Access (JPA).	**Collaborations:** ▪ Course

4.2.2.2 REPOSITORY

The repository package classes are documented in tables Table 4.4 to Table 4.6.

Table 4.4: CRC Card for the InstructorRepository class

Class Name: {Interface} IntstructorRepository	
Responsibilities: ▪ Extends the CrudRepository Spring class which provides access to the database. ▪ Defines a method to return an instructor object from a string username.	**Collaborations:** ▪ model.Instructor

Table 4.5: CRC Card for the CourseRepository class

Class Name: {Interface} CourseRepository	
Responsibilities: ▪ Extends the CrudRepository Spring class which provides access to the database. ▪ Defines a method which returns a list of courses by a given instructor instance.	**Collaborations:** ▪ model.Instructor ▪ model.Course

Table 4.6: CRC Card for the StudentRegistrationRepository class

Class Name: {Interface} StudentRegistrationRepository	
Responsibilities: ▪ Extends the CrudRepository Spring class which provides access to the database. ▪ Defines a method which returns a list of student registrations by a given course instance.	**Collaborations:** ▪ model.Course ▪ model.StudentRegistration

4.2.2.3 DATA ACCESS OBJECTS (DAO)

The dao package classes are documented in tables Table 4.7 to Table 4.12.

Table 4.7: CRC Card for the InstructorDAO class

Class Name: {Interface} InstructorDAO	
Responsibilities: ▪ Constitutes an interface for potential implementations of the dao layer. ▪ Provides following methods: o Check if an instructor exists in the database by username. o Retrieve instructor password by username. o Register instructor. o Retrieve instructor instance by username.	**Collaborations:** ▪ model.Instructor

Table 4.8: CRC Card for the InstructorDAOImpl class

Class Name: InstructorDAOImpl	
Responsibilities: ▪ The is a concrete implementation of the InstructorDAO interface. This implementation is using the Spring JPA for database access. ▪ Provides a concrete implementation for each interface method.	**Collaborations:** ▪ model.Instructor. ▪ repository.InstructorRepository

Table 4.9: CRC Card for the CourseDAO class

Class Name: {Interface} CourseDAO	
Responsibilities: ▪ Constitutes an interface for potential implementations of the dao layer. ▪ Provides following methods: o Retrieve a list of courses by instructor login. o Update a course. o Save a course. o Delete a course. o Retrieve a course by id.	**Collaborations:** ▪ model.Course

Table 4.10: CRC Card for the CourseDAOImpl class

Class Name: CourseDAOImpl	
Responsibilities: ▪ The is a concrete implementation of the CourseDAO interface. This implementation is using the Spring JPA for database access. ▪ Provides a concrete implementation for each interface method.	**Collaborations:** ▪ model.Course ▪ model.Instructor ▪ repository.CourseRepository

Table 4.11: CRC Card for the StudentRegistrationDAO class

Class Name: {Interface} StudentRegistrationDAO	
Responsibilities: ▪ Constitutes an interface for potential implementations of the dao layer. ▪ Provides following methods: o Retrieve a list of student registrations by course id. o Update a student registration. o Save a student registration o Delete a student registration o Retrieve a student registration by id.	**Collaborations:** ▪ model.StudentRegistration

Table 4.12: CRC Card for the StudentRegistrationDAOImpl class

Class Name: StudentRegistrationDAOImpl	
Responsibilities: ▪ The is a concrete implementation of the StudentRegistrationDAO interface. This implementation is using the Spring JPA for database access. ▪ Provides a concrete implementation for each interface method.	**Collaborations:** ▪ model.Course ▪ model.StudentRegistration ▪ repository.StudentRegistrationRepository

4.2.2.4 SERVICE

The service package classes are documented in tables Table 4.13 to Table 4.29.

Table 4.13: CRC Card for the InstructorService class

Class Name: {Interface} InstructorService	
Responsibilities: ▪ Constitutes an interface for potential implementations of the service layer. ▪ Provides following methods: o Check if an instructor exists by username. o Retrieve instructor password by username. o Register instructor. o Retrieve instructor instance by username.	**Collaborations:** ▪ model.Instructor

Table 4.14: CRC Card for the InstructorServiceImpl class

Class Name: InstructorServiceImpl	
Responsibilities: ▪ The is a concrete implementation of the InstructorService interface. ▪ Provides a concrete implementation for each interface method. All the provided operations correspond directly to respective DAO operations.	**Collaborations:** ▪ model.Instructor ▪ dao.InstructorDAO

Table 4.15: CRC Card for the CourseService class

Class Name: {Interface} CourseService	
Responsibilities: ▪ Constitutes an interface for potential implementations of the service layer. ▪ Provides following methods: o Retrieve a list of courses by instructor login. o Update a course.	**Collaborations:** ▪ model.Course

○ Save a course. ○ Delete a course. ○ Retrieve a course by id. ○ Return a map of course statistic name – course statistic value ○ Return a list of statistic strategies ○ Set a list of statistic strategies	

Table 4.16: CRC Card for the CourseServiceImpl class

Class Name: CourseServiceImpl	
Responsibilities: ▪ The is a concrete implementation of the CourseService interface. ▪ Provides a concrete implementation for each interface method. Some of the provided operations correspond directly to respective DAO operations.	**Collaborations:** ▪ dao.CourseDAO ▪ dao.StudentRegistrationDAO ▪ model.Course ▪ model.StudentRegistration

Table 4.17: CRC Card for the StudentRegistrationService class

Class Name: {Interface} StudentRegistrationService	
Responsibilities: ▪ Constitutes an interface for potential implementations of the service layer. ▪ Provides following methods: ○ Retrieve a list of student registrations by course id. ○ Update a student registration. ○ Save a student registration ○ Delete a student registration ○ Retrieve a student registration by id.	**Collaborations:** ▪ model.StudentRegistration

Table 4.18: CRC Card for the StudentRegistrationImpl class

Class Name: StudentRegistrationImpl	
Responsibilities: ▪ The is a concrete implementation of the StudentRegistrationService interface. ▪ Provides a concrete implementation for each interface method. All the provided operations correspond directly to respective DAO operations.	**Collaborations:** ▪ dao.StudentRegistrationDAO ▪ model.StudentRegistration

Table 4.19: CRC Card for the StatisticStrategy class

Class Name: {Interface} StatisticStrategy	
Responsibilities: ▪ Constitutes an interface for potential implementations of different statistics. ▪ Provides following methods: o Calculate a statistic for a given list of student registrations o Return the statistic name.	**Collaborations:** ▪ model.StudentRegistration

Table 4.20: CRC Card for the TemplateStatisticStrategy class

Class Name: {Abstract} TemplateStatisticStrategy	
Responsibilities: ▪ Constitutes an abstract class for all the classes that implement the statistic interface. ▪ It accommodates the Template Method GoF pattern. ▪ Provides abstract methods for: o The actual statistic calculation. o Return the statistic name. ▪ Provides following concrete methods: o Calculate statistic (interface implementation). o Prepare data set.	**Collaborations:** ▪ model.StudentRegistration

Table 4.21: CRC Card for the MinStatisticStrategy class

Class Name: MinStatisticStrategy	
Responsibilities: ▪ Extends the TemplateStatisticStrategy. ▪ Performs the actual min statistic calculation. ▪ Returns the statistic name.	**Collaborations:** ▪ none

Table 4.22: CRC Card for the MaxStatisticStrategy class

Class Name: MaxStatisticStrategy	
Responsibilities: ▪ Extends the TemplateStatisticStrategy. ▪ Performs the actual max statistic calculation. ▪ Returns the statistic name.	**Collaborations:** ▪ none

Table 4.23: CRC Card for the MeanStatisticStrategy class

Class Name: MeanStatisticStrategy	
Responsibilities: ▪ Extends the TemplateStatisticStrategy. ▪ Performs the actual mean statistic calculation. ▪ Returns the statistic name.	**Collaborations:** ▪ none

Table 4.24: CRC Card for the StandardDeviationStatisticStrategy class

Class Name: StandardDeviationStatisticStrategy	
Responsibilities: ▪ Extends the TemplateStatisticStrategy. ▪ Performs the actual standard deviation statistic calculation. ▪ Returns the statistic name.	**Collaborations:** ▪ none

Table 4.25: CRC Card for the VarianceStatisticStrategy class

Class Name: VarianceStatisticStrategy	
Responsibilities: ▪ Extends the TemplateStatisticStrategy. ▪ Performs the actual variance statistic calculation. ▪ Returns the statistic name.	**Collaborations:** ▪ none

Table 4.26: CRC Card for the NinetyFifthPercentileStatisticStrategy class

Class Name: NinetyFifthPercentileStatisticStrategy	
Responsibilities: ▪ Extends the TemplateStatisticStrategy. ▪ Performs the actual 95th percentile statistic calculation. ▪ Returns the statistic name.	**Collaborations:** ▪ none

Table 4.27: CRC Card for the SkewnessStatisticStrategy class

Class Name: SkewnessStatisticStrategy	
Responsibilities: ▪ Extends the TemplateStatisticStrategy. ▪ Performs the actual skewness statistic calculation. ▪ Returns the statistic name.	**Collaborations:** ▪ none

Table 4.28: CRC Card for the KurtosisStatisticStrategy class

Class Name: KurtosisStatisticStrategy	
Responsibilities: ▪ Extends the TemplateStatisticStrategy. ▪ Performs the actual kurtosis statistic calculation. ▪ Returns the statistic name.	**Collaborations:** ▪ none

Table 4.29: CRC Card for the MedianStatisticStrategy class

Class Name: MedianStatisticStrategy	
Responsibilities: ▪ Extends the TemplateStatisticStrategy. ▪ Performs the actual median statistic calculation. ▪ Returns the statistic name.	**Collaborations:** ▪ none

4.2.2.5 CONTROLLER

The controller package classes are documented in tables Table 4.30 to Table 4.34.

Table 4.30: CRC Card for the LoginController class

Class Name: LoginController	
Responsibilities: ▪ Facilitates the user interaction with the login page. ▪ Performs checks of credentials and logs in a registered instructor. ▪ Directs to the register page if the user chooses to register.	**Collaborations:** ▪ model.Course ▪ service.CourseService ▪ service.InstructorService

Table 4.31: CRC Card for the RegisterController class

Class Name: RegisterController	
Responsibilities: ▪ Facilitates the user interaction with the register page. ▪ Performs checks of credentials and registers a new instructor. ▪ Directs to the login page upon successful registration or cancel.	**Collaborations:** ▪ model.Instructor ▪ service.InstructorService

Table 4.32: CRC Card for the CourseController class

Class Name: CourseController	
Responsibilities: ▪ Facilitates the user interaction with the courses page. ▪ Provides methods which perform the following operations: o Show the add new course form. o Add new course. o Cancel add new course. o Show student registrations. o Show course edit form. o Edit course. o Cancel the course edit. o Delete course. o Log out. o Show help page. o Go back from help page.	**Collaborations:** ▪ CourseController ▪ model.Instructor ▪ model.StudentRegistration ▪ service.CourseService ▪ service.InstructorService ▪ service.StudentRegistrationService

Table 4.33: CRC Card for the StudentRegistrationController class

Class Name: StudentRegistrationController	
Responsibilities: ▪ Facilitates the user interaction with the student registrations page. ▪ Provides methods which perform the following operations: o Show the add new student registration form. o Add new student registration. o Cancel add new student registration. o Show student registration edit form.	**Collaborations:** ▪ model.Course; ▪ model.StudentRegistration; ▪ service.CourseService; ▪ service.KurtosisStatisticStrategy; ▪ service.MaxStatisticStrategy; ▪ service.MeanStatisticStrategy; ▪ service.MedianStatisticStrategy; ▪ service.MinStatisticStrategy; ▪ service.NinetyFifthPercentileStatisticStrategy; ▪ service.SkewnessStatisticStrategy;

○ Edit student registration. ○ Cancel the student registration edit. ○ Delete student registration. ○ Go back to the courses page. ○ Calculate final grades for all students. ○ Calculate statistics and show statistics page. ○ Show help page. ○ Go back from help page.	▪ service.StandardDeviationStatisticStrategy; ▪ service.StatisticStrategy; ▪ service.StudentRegistrationService; ▪ service.VarianceStatisticStrategy

Table 4.34: CRC Card for the StatisticsController class

Class Name: StatisticsController	
Responsibilities: ▪ Facilitates the user interaction with the statistics page. ▪ Directs back to the student registrations page	**Collaborations:** ▪ model.StudentRegistration ▪ service.StudentRegistrationService

5 Testing

For each layer of the application several tests have been developed for each class involved. To ensure each layer is tested independently of other layers, the Mockito framework has been used. The Mockito framework "mocks" classes of different layers used in a certain layer to isolate the logic of methods under test.

The following sections present the tests developed in tabular mode.

5.1 Model

For the domain model layer of the application a set of unit tests have been developed for each of the involved classes.

The tests developed for the "Instructor" class are presented in Table 5.1.

Table 5.1: Instructor class tests

Test ID	Test Name
I_M1_01	testSetAndGetUsernameHappyDay
I_M1_02	testSetAndGetUsernameEmptyString
I_M2_01	testSetAndGetPasswordHappyDay
I_M2_02	testSetAndGetPasswordEmptyString

The tests developed for the "Course" class are presented in Table 5.2.

Table 5.2: Course class tests

Test ID	Test Name
C_M1_01	testSetAndGetIdHappyDay
C_M1_02	testSetAndGetIdZero
C_M1_03	testSetAndGetIdNegative
C_M2_01	testSetAndGetNameHappyDay
C_M2_02	testSetAndGetNameEmptyString
C_M3_01	testSetAndGetDescriptionHappyDay
C_M3_02	testSetAndGetDescriptionEmptyString
C_M4_01	testSetAndGetSyllabusHappyDay
C_M4_02	testSetAndGetSyllabusEmptyString
C_M5_01	testSetAndGetInstructorHappyDay
C_M5_02	testSetAndGetInstructorNull
C_M6_01	testSetAndGetYearHappyDay
C_M6_02	testSetAndGetYearLarge
C_M6_03	testSetAndGetYearZero
C_M6_04	testSetAndGetYearNegative
C_M7_01	testSetAndGetSemesterHappyDay

C_M7_02	testSetAndGetSemesterLarge
C_M7_03	testSetAndGetSemesterZero
C_M7_04	testSetAndGetSemesterNegative
C_M8_01	testSetAndGetExamWeightHappyDay
C_M8_02	testSetAndGetExamWeightLarge
C_M8_03	testSetAndGetExamWeightZero
C_M8_04	testSetAndGetExamWeightNegative

The tests developed for the "StudentRegistration" class are presented in Table 5.3.

Table 5.3: StudentRegistration class tests

Test ID	Test Name
SR_M1_01	testSetAndGetIdHappyDay
SR_M2_01	testSetAndGetNamedHappyDay
SR_M2_02	testSetAndGetNamedEmptyString
SR_M3_01	testSetAndGetYearOfStudiesHappyDay
SR_M3_02	testSetAndGetYearOfStudiesLarge
SR_M3_03	testSetAndGetYearOfStudiesZero
SR_M3_04	testSetAndGetYearOfStudiesNegative
SR_M4_01	testSetAndGetCourseHappyDay
SR_M4_02	testSetAndGetCourseNull
SR_M5_01	testSetAndGetSemesterHappyDay
SR_M5_02	testSetAndGetSemesterLarge
SR_M5_03	testSetAndGetSemesterZero
SR_M5_04	testSetAndGetSemesterNegative
SR_M6_01	testSetAndGetYearOfRegistrationHappyDay
SR_M6_02	testSetAndGetYearOfRegistrationZero
SR_M6_03	testSetAndGetYearOfRegistrationNegative
SR_M7_01	testSetAndGetProjectGradeHappyDay
SR_M7_02	testSetAndGetProjectGradeLarge
SR_M7_03	testSetAndGetProjectGradeZero
SR_M7_04	testSetAndGetProjectGradeNegative
SR_M8_01	testSetAndGetExamGradeHappyDay
SR_M8_02	testSetAndGetExamGradeLarge
SR_M8_03	testSetAndGetExamGradeZero
SR_M8_04	testSetAndGetExamGradeNegative
SR_M9_01	testSetAndGetGradeHappyDay
SR_M9_02	testSetAndGetGradeLarge
SR_M9_03	testSetAndGetGradeZero
SR_M9_04	testSetAndGetGradeNegative

5.2 Data Access Objects (DAO)

For the dao layer of the application a set of integration tests have been developed for each of the involved classes. To test the dao layer methods, the underneath layer classes (Repositories) have been mocked with Mockito.

It should be noted here that void methods which interact directly with the repository classes and in effect the MySQL database, have not been tested. It was not understood how they can be tested without involving the database or changing the method return type in the interface provided.

The tests developed for the "InstructorDAOImpl" class are presented in Table 5.4.

Table 5.4: InstructorDAOImpl class tests

Test ID	Test Name
ID_M1_01	testExistsInstructorHappyDay
ID_M1_02	testExistsInstructorNotPresent
ID_M2_01	testGetInstructorPasswordHappyDay
ID_M2_02	testGetInstructorPasswordNotPresent
ID_M3_01	testGetInstructorHappyDay
ID_M3_02	testGetInstructorNotPresent

The tests developed for the "CourseDAOImpl" class are presented in Table 5.5.

Table 5.5: CourseDAOImpl class tests

Test ID	Test Name
CD_M1_01	testFindCourseByInstructorLoginHappyDay
CD_M1_02	testFindCourseByInstructorLoginEmptyList
CD_M2_01	testGetCourseHappyDay
CD_M2_02	testGetCourseNotPresent

The tests developed for the "StudentRegistrationDAOImpl" class are presented in Table 5.6.

Table 5.6: StudentRegistrationDAOImpl class tests

Test ID	Test Name
SRD_M1_01	testFindRegistrationByCourseIdHappyDay
SRD_M1_02	testFindRegistrationByCourseIdEmptyList
SRD_M2_01	testGetStudentRegistrationHappyDay
SRD_M2_02	testGetStudentRegistrationNotPresent

5.3 Service

For the service layer of the application a set of integration tests have been developed for each of the involved classes. To test the service layer methods, the underneath layer classes (dao) have been mocked with Mockito. Similarly, void method could not be tested.

The tests developed for the “InstructorServiceImpl” class are presented in Table 5.7.

Table 5.7: InstructorServiceImpl class tests

Test ID	Test Name
IS_M1_01	testExistsInstructorHappyDay
IS_M2_01	testGetInstructorPasswordHappyDay
IS_M3_01	testGetInstructorHappyDay

The tests developed for the “CourseServiceImpl” class are presented in Table 5.8.

Table 5.8: CourseServiceImpl class tests

Test ID	Test Name
CS_M1_01	testfindCourseByInstructorLoginHappyDay
CS_M1_02	testfindCourseByInstructorLoginEmptyList
CS_M2_01	testGetCourseHappyDay
CS_M3_01	testGetCourseStatisticsHappyDay
CS_M3_02	testGetCourseStatisticsEmptyList
CS_M4_01	testSetAndGetStatCalculationStrategiesHappyDay
CS_M4_02	testSetAndGetStatCalculationStrategiesEmptyList

The tests developed for the “StudentRegistrationServiceImpl” class are presented in Table 5.9.

Table 5.9: StudentRegistrationServiceImpl class tests

Test ID	Test Name
SRS_M1_01	testFindRegistrationByCourseIdHappyDay
SRS_M1_02	testFindRegistrationByCourseIdEmptyList
SRS_M2_01	testGetStudentRegistrationHappyDay

5.4 Controller

For the controller layer of the application a set of acceptance tests have been developed for each of the involved classes. To test the controller layer methods, the underneath layer classes (service) have been mocked with Mockito.

The tests developed for the “LoginController” class are presented in Table 5.10.

Table 5.10: LoginController class tests

Test ID	Test Name
LC_M1_01	testLoginHappyDay
LC_M1_02	testLoginBlankUserName
LC_M1_03	testLoginNonExistentInstructor
LC_M1_04	testLoginWrongPassword
LC_M2_01	testRegisterHappyDay

The tests developed for the “RegisterController” class are presented in Table 5.11.

Table 5.11: RegisterController class tests

Test ID	Test Name
RC_M1_01	testRegisterInstructorHappyDay
RC_M1_02	testRegisterInstructorInstructorExists
RC_M1_03	testRegisterInstructorEmptyPassword
RC_M1_04	testRegisterInstructorPasswordMismatch
RC_M2_01	testCancelRegisterHappyDay

The tests developed for the “CourseController” class are presented in Table 5.12.

Table 5.12: CourseController class tests

Test ID	Test Name
CC_M1_01	testShowAddCourseForm
CC_M2_01	testAddNewCourseHappyDay
CC_M2_02	testAddNewCourseIsBlank
CC_M2_03	testAddNewCourseInvalidYear
CC_M2_04	testAddNewCourseInvalidSemester
CC_M2_05	testAddNewCourseInvalidExamWeight
CC_M3_01	testCancelAddHappyDay
CC_M4_01	testShowStudentRegistrationsHappyDay
CC_M4_02	testShowEditCourseFormHappyDay
CC_M5_01	testEditCourseHappyDay
CC_M5_02	testEdiCourseIsBlank
CC_M5_03	testEditCourseInvalidYear
CC_M5_04	testEditCourseInvalidSemester
CC_M5_05	testEditCourseInvalidExamWeight
CC_M6_01	testDeleteCourseHappyDay
CC_M7_01	testCancelCourseEditHappyDay
CC_M8_01	testlogOutHappyDay
CC_M9_01	testShowCourseHelpHappyDay
CC_M10_01	testGoBackFromHelpHappyDay

The tests developed for the “StudentRegistrationController” class are presented in Table 5.13.

Table 5.13: StudentRegistrationController class tests

Test ID	Test Name
SRC_M1_01	testShowAddStudentForm
SRC_M2_01	testAddNewStudentRegistrationHappyDay
SRC_M2_02	testAddNewStudentRegistrationIsBlank
SRC_M2_03	testAddNewStudentRegistrationInvalidYearOfStudies
SRC_M2_04	testAddNewStudentRegistrationInvalidSemester
SRC_M2_05	testAddNewStudentRegistrationInvalidYearOfRegistration
SRC_M2_06	testAddNewStudentRegistrationInvalidProjectGrade
SRC_M2_07	testAddNewStudentRegistrationInvalidExamGrade
SRC_M3_01	testCancelAddNewStudentHappyDay
SRC_M4_01	testShowEditStudentFormHappyDay
SRC_M5_01	testEditStudentRegistrationHappyDay
SRC_M5_02	testEditStudentRegistrationIsBlank
SRC_M5_03	testEditStudentRegistrationInvalidYearOfStudies
SRC_M5_04	testEditStudentRegistrationInvalidSemester
SRC_M5_05	testEditStudentRegistrationInvalidYearOfRegistration
SRC_M5_06	testEditStudentRegistrationInvalidProjectGrade
SRC_M5_07	testEditStudentRegistrationInvalidExamGrade
SRC_M6_01	testCancelStudentEditHappyDay
SRC_M7_01	testDeleteStudentHappyDay
SRC_M8_01	testGoBackToCoursesHappyDay
SRC_M9_01	testCalculateFinalGradesHappyDay
SRC_M10_01	testCalculateStatsHappyDay
SRC_M11_01	testShowRegistrationsHelpHappyDay
SRC_M12_01	testBackFromRegistrationsHelpHappyDay

The tests developed for the “StatisticsController” class are presented in Table 5.14.

Table 5.14: StatisticsController class tests

Test ID	Test Name
SC_M1_01	testGoBackToRegistrations

6 Assumptions / Limitations

During the application development process, a few assumptions were made. Some of them could be seen as limitations of the current release of the application and actions for a future release. These assumptions/limitations are discussed in the following sections.

6.1 User stories 9 & 10 combined

The user stories 9 (update students' information) and 10 (register student project and final exam grades) are fulfilled via the same form i.e., the "Add New Student" and "Edit Student" forms.

6.2 Table Column Widths

The web application presents the required information in tables. These tables currently support columns with fixed widths. The widths were derived based on the developer's assumption on the expected input information.

6.3 Course Project – Exam Weights

One of the application's functional requirements is to calculate the final grade for each course managed. To achieve this, the instructor must provide project and exam weights. The application currently offers an input field only for the exam weight. The project weight is calculated as 1 – exam weight.

Furthermore, the exam/project weights are not displayed in the courses pages but are hidden in the background. The instructor must provide a value for the exam weight when creating a new course and can update this weight when editing the course but can not directly see these weights on the respective table.