SOFTWARE ENGINEERING
OVERALL PROJECT REPORT:

Developing a Courses Management Web App with Spring Boot

TEAM

Antoniou Christodoulos	AM: 2641
Tsiouri Angeliki	AM: 3354

Zengineers

TABLE OF CONTENTS

SUMMARY

The objective of the project was to develop a Web App that allows an instructor to manage the grading of the courses that he teaches. The objective was to be fulfilled using a Scrum approach to resolve organizational issues and technologies such as the Spring Boot Framework and MySQL to satisfy functional requirements. Non-functional requirements were specified in the form of design quality and best coding practices. Incomplete class diagrams describing the application architecture was the starting point. Testing was also among the requirements.

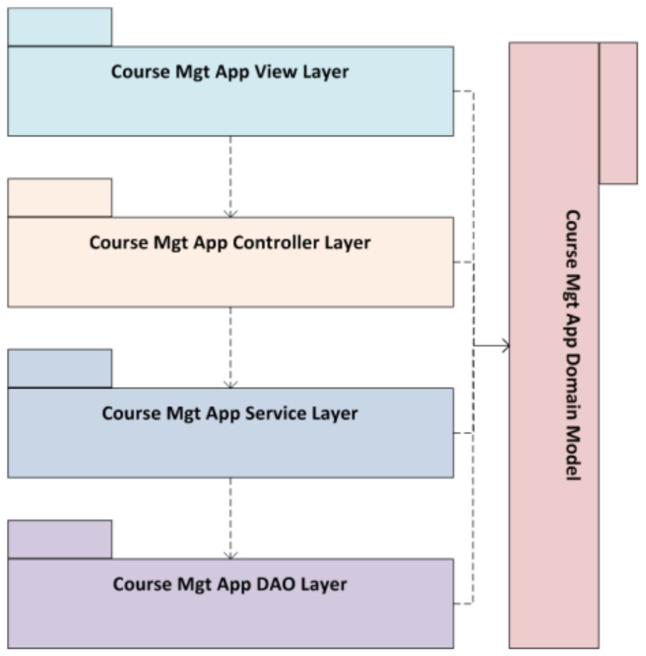

Course Mgt App View Layer
Course Mgt App Controller Layer
Course Mgt App Service Layer
Course Mgt App DAO Layer
Course Mgt App Domain Model

FIGURE 1 APPLICATION ARCHITECTURE.

This document provides information concerning the 4 sprints of development for the Courses Web App. The development process has also been documented under the description of each commit on GitHub.

TECHNOLOGIES

This section contains a brief overview of the technologies used:

- **MySQL** & **MySQL Workbench** as a database and DBMS respectively.
- **GitHub** for version control. The following link points to the project repo with all the project files and detailed information about commits and pull requests. [https://github.com/Zengineers/courses-web-app]
- **Trello** for project management and team organization. The following link points to the project board which contains cards with the user stories and any other task regarding the implementation of the project as well as various useful information in the form of links we came across during the development process. [https://trello.com/b/aYWeaLxz/zengineers-coursesmanagementapp]
- **Figma** as an application wireframe design tool. All the application's pages were first designed here along with a rough sketch of their navigation functionality. Later, the Figma wireframe was used as a guideline to design the actual application using CSS and HTML. The following link points to the wireframe. A presentation can be instantiated by clicking on the present button on the top right. [https://www.figma.com/file/PYsLIdOQx0acj7uvoIks2C/Courses-Management-App-Wireframe?node-id=0%3A1]
- **Spring boot** as a general framework for creating the application with **Java 11** as the programming language and **Maven** as a build tool. Some notable dependencies were **Thymeleaf** as a template engine, **Spring Data JPA** and **MySQL Driver** to implement the connection with the MySQL database and **Spring Boot DevTools** to auto-reload the project when the source code is changed while the application is running.
- **Eclipse** as an IDE.
- **JUnit 5** for testing.
- **ObjectAid** (Eclipse plugin) to generate UML package and class diagrams.
- **Discord** as a communication tool.

SCRUM TEAM & SPRINTS

The Scrum Team:

Product Owner	Tsiouri Angeliki
Scrum Master	Antoniou Christodoulos
Development Team	Antoniou Christodoulos, Tsiouri Angeliki

Sprints:

Sprint No	Begin Date	End Date	Number of weeks	Tasks
0	25 Feb	16 Mar	3	Preparation
1	17 Mar	10 Apr	3.5	User stories (US1 till US5)
2	25 Apr	6 May	2	User stories (US6 till US9)
3	8 May	18 May	1.5	User stories (US10, US10, US6-BONUS), Presentation

Preparation refers to tasks revolving around familiarization with new technologies such as Spring, Spring boot, MySQL Workbench, as well as building mock projects and a functional application prototype. The following preparation tasks were completed in the first sprint (respective cards may also be viewed on Trello):

- Setup Figma Wireframe
- Setup MySQL
- Study Spring, Spring Boot, Thymeleaf
- Setup Application Architecture
- Setup Functional Prototype

Presentation refers to tasks revolving around the documentation and presentation of the application. The following presentation tasks were completed in the 4th sprint:

- Write report (includes: CRC cards, UML package and class diagrams, export database schema)
- Write GitHub Readme
- Prepare Demo Video

Regarding the implementation of the user stories, each one was split into a checklist of 4 items:

- UI Design – HTML Styling
- Functionality Implementation
- Test (Validation)
- Use case

Once all 4 items of the checklist were completed, the user story was considered to be done.

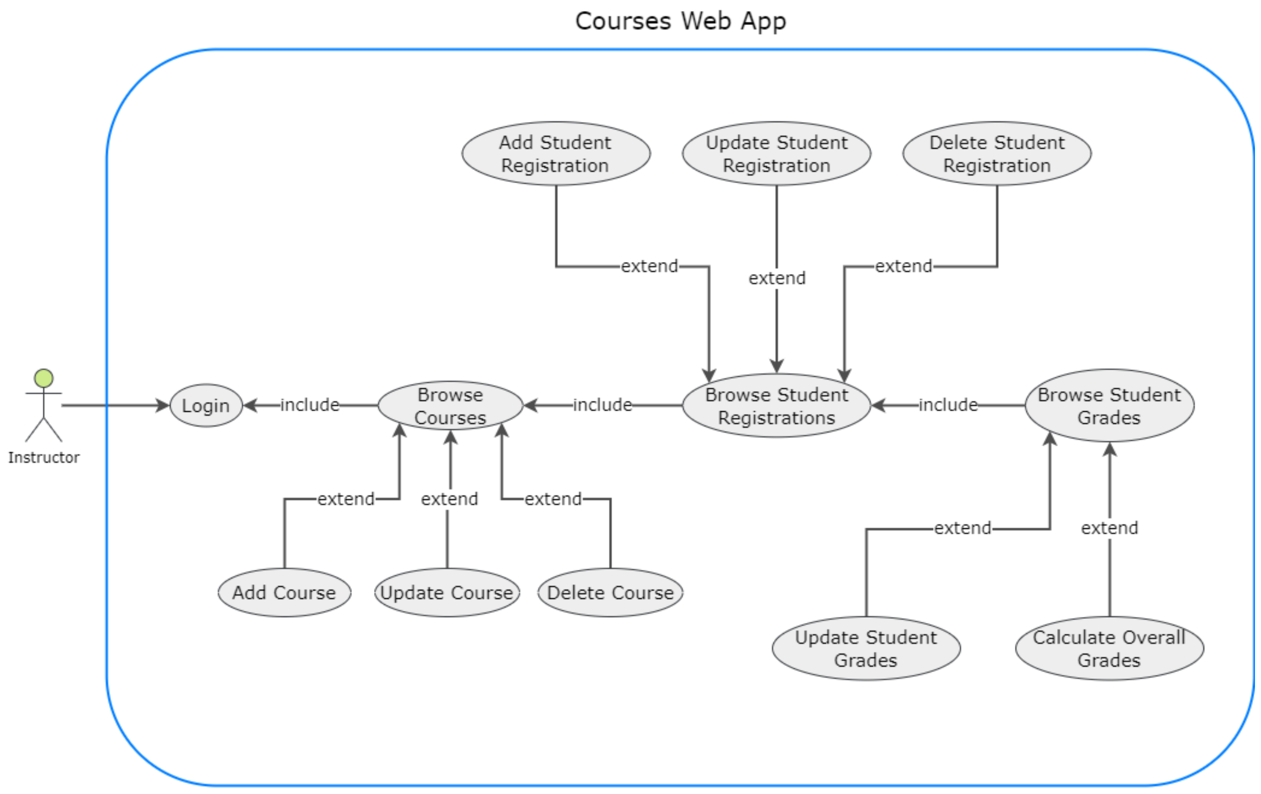
BACKLOG – USER STORIES

ID	**AS A <User Type>**	**I WANT <An Action>**	**SO THAT <A Benefit/Value>**
US1	Instructor	To login to the application with my username and password.	To safely manage the courses that I teach.
US2	Instructor	To browse the list of my courses.	To manage the courses' descriptions and the students' grades.
US3	Instructor	To add a course in the list, by giving information like the course id, name, syllabus, year, semester, etc.	To populate the list of the courses with new ones.
US4	Instructor	To remove a course from the list.	Clean up the list of courses.
US5	Instructor	To update the description of a course.	Correct possible mistakes and keep the information up to date with the current situation.
US6	Instructor	To browse the list of students that enrolled to a particular course.	To manage the students that enrolled in the course.
US7	Instructor	To add a student to the list of a particular course, by giving information like the student id, name, year of registration, semester, etc.	To populate the list with the students that enrolled in the course.

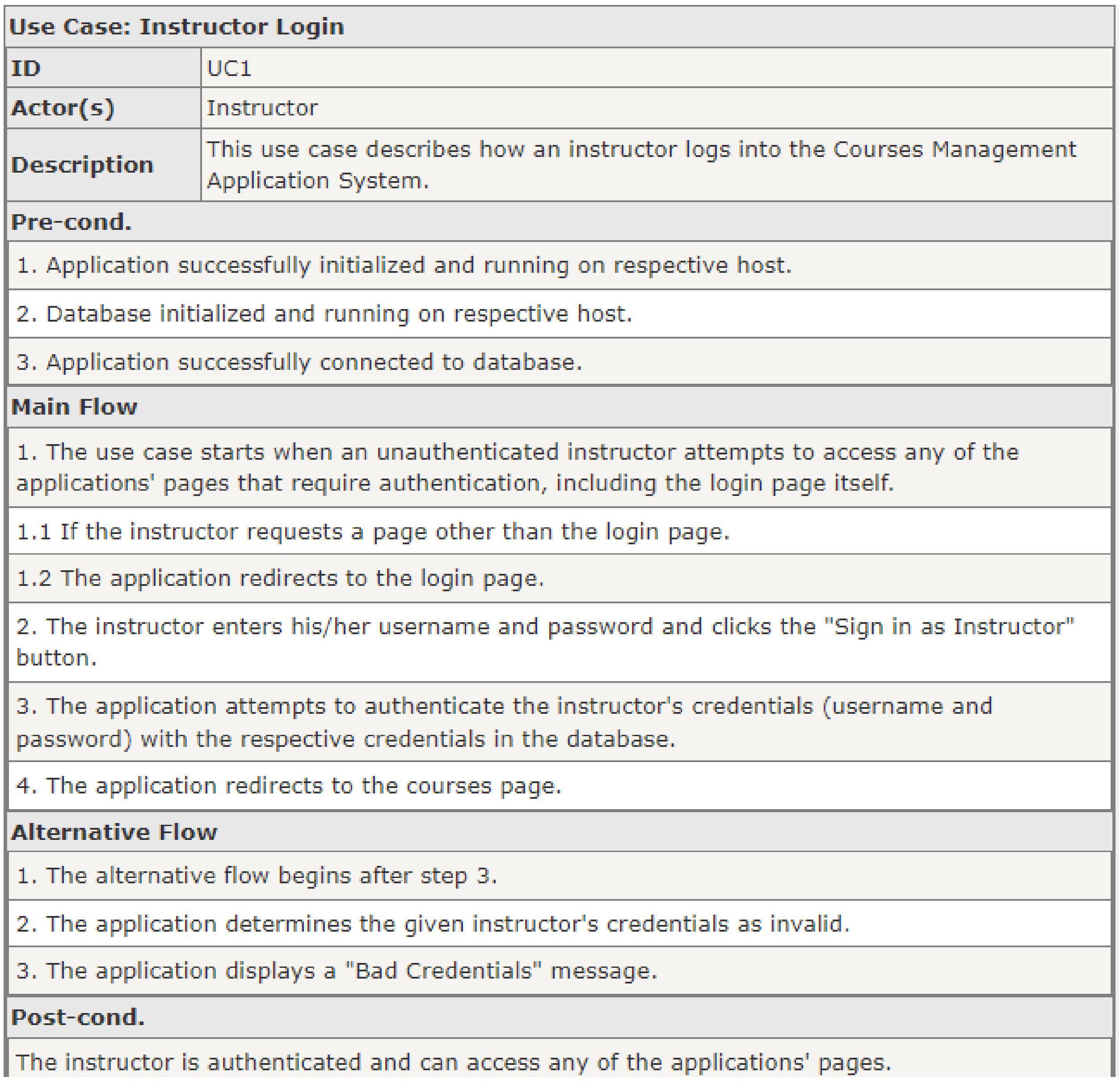
US8	Instructor	To remove a student from the list of a particular course.	To deal with students that resigned from the course.
US9	Instructor	To update students' information (id, name, year of studies, etc.).	Correct possible mistakes and keep the information up to date with the current situation.
US6-BONUS	Instructor	To browse the list of student grades for students that enrolled in a particular course.	To manage the grading of the students that enrolled in the course.
US10	Instructor	To register the grades of the student in the final exam and the project of the course.	To manage the grading of the students that enrolled in the course.
US11	Instructor	To calculate the overall grades of the students that enrolled in a particular course with respected to a weighted average.	To manage the grading of the students that enrolled in the course.
US12	Instructor	To calculate descriptive statistics about the students' grades in a particular course, including min, max, mean, standard deviation, variance, percentiles, skewness, kurtosis, median.	Study the distribution of the students' grades in the course.

USE CASES

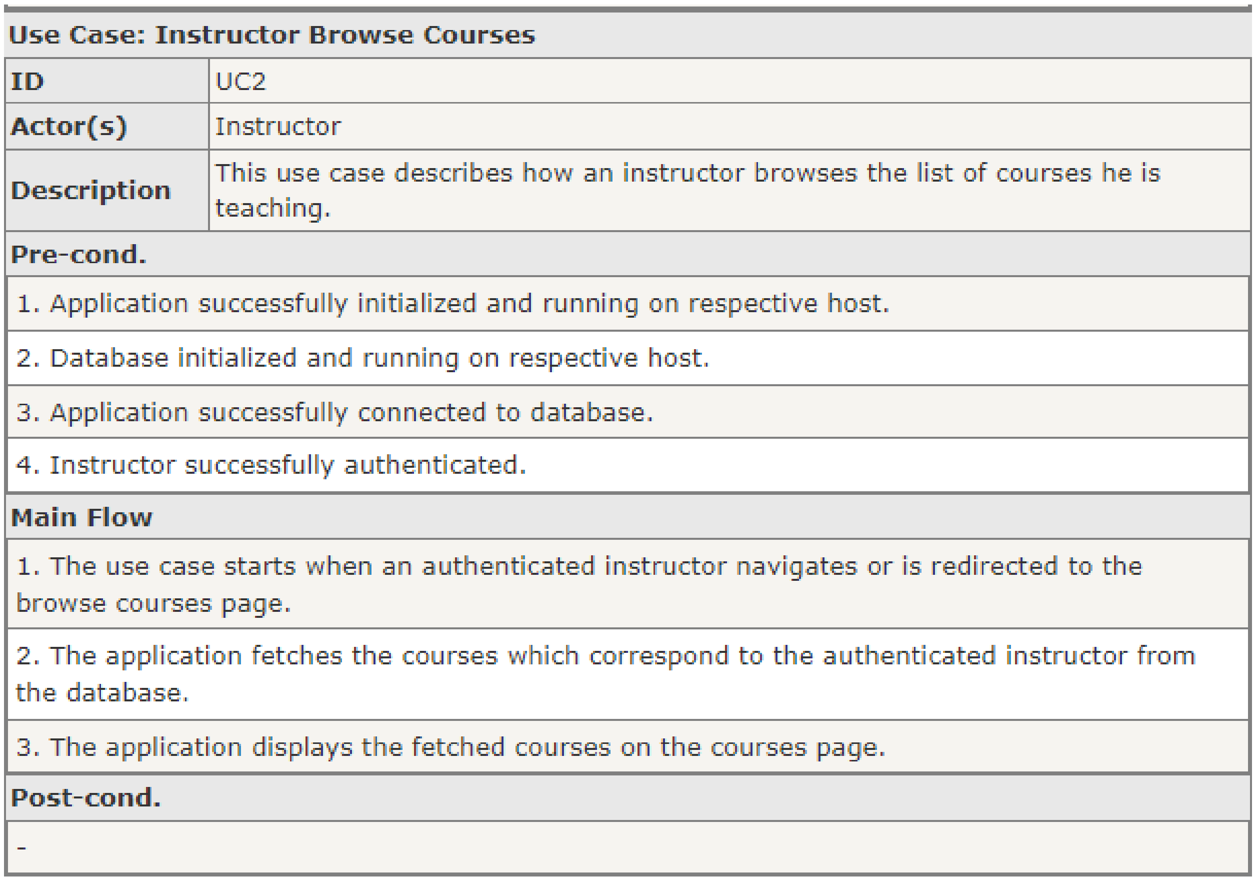
Use case UML diagram describing the general functionality of the application:



And the use cases in detail:

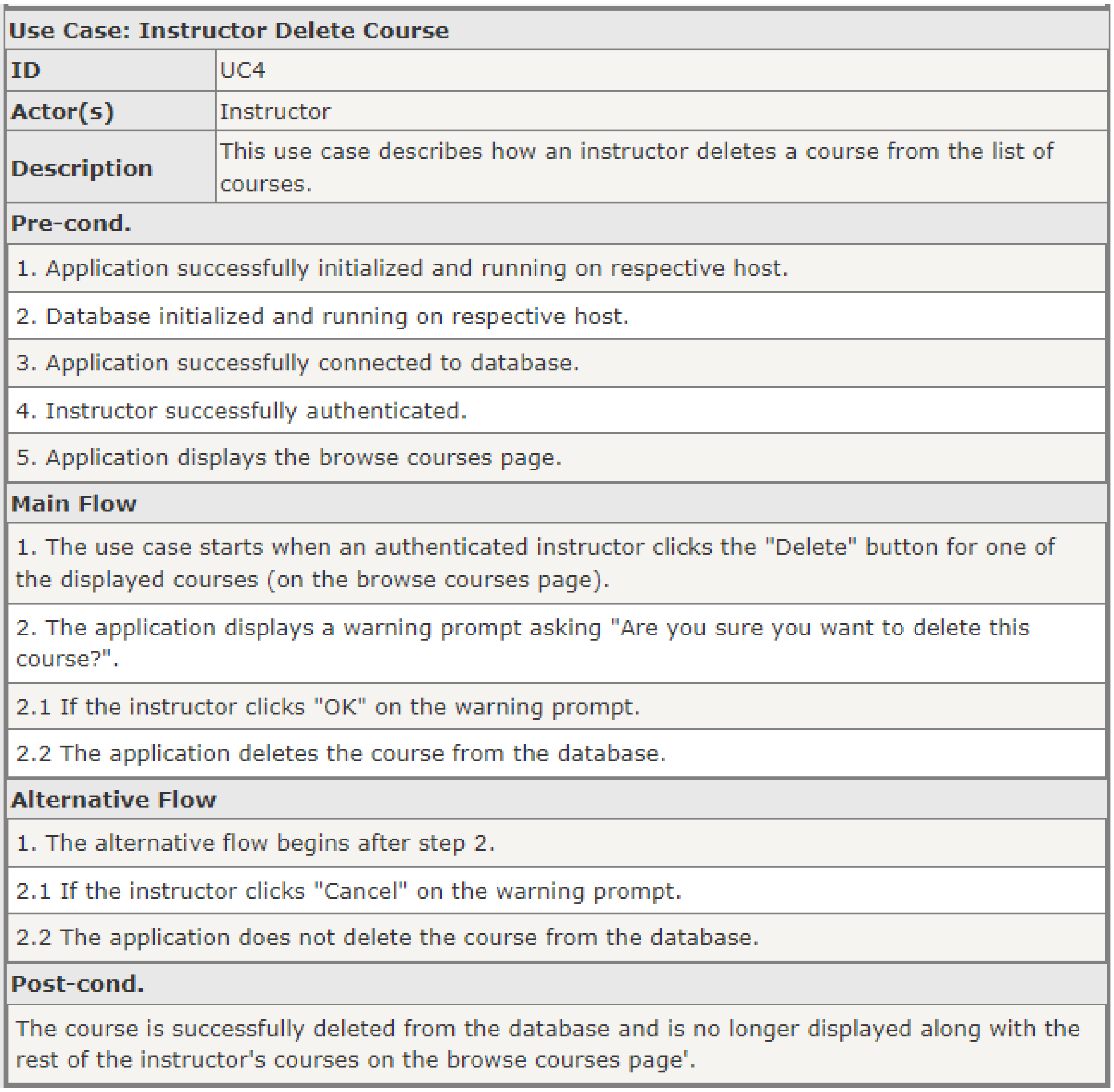


Use Case: Instructor Login	
ID	UC1
Actor(s)	Instructor
Description	This use case describes how an instructor logs into the Courses Management Application System.
Pre-cond.	
1. Application successfully initialized and running on respective host.	
2. Database initialized and running on respective host.	
3. Application successfully connected to database.	
Main Flow	
1. The use case starts when an unauthenticated instructor attempts to access any of the applications' pages that require authentication, including the login page itself.	
1.1 If the instructor requests a page other than the login page.	
1.2 The application redirects to the login page.	
2. The instructor enters his/her username and password and clicks the "Sign in as Instructor" button.	
3. The application attempts to authenticate the instructor's credentials (username and password) with the respective credentials in the database.	
4. The application redirects to the courses page.	
Alternative Flow	
1. The alternative flow begins after step 3.	
2. The application determines the given instructor's credentials as invalid.	
3. The application displays a "Bad Credentials" message.	
Post-cond.	
The instructor is authenticated and can access any of the applications' pages.	

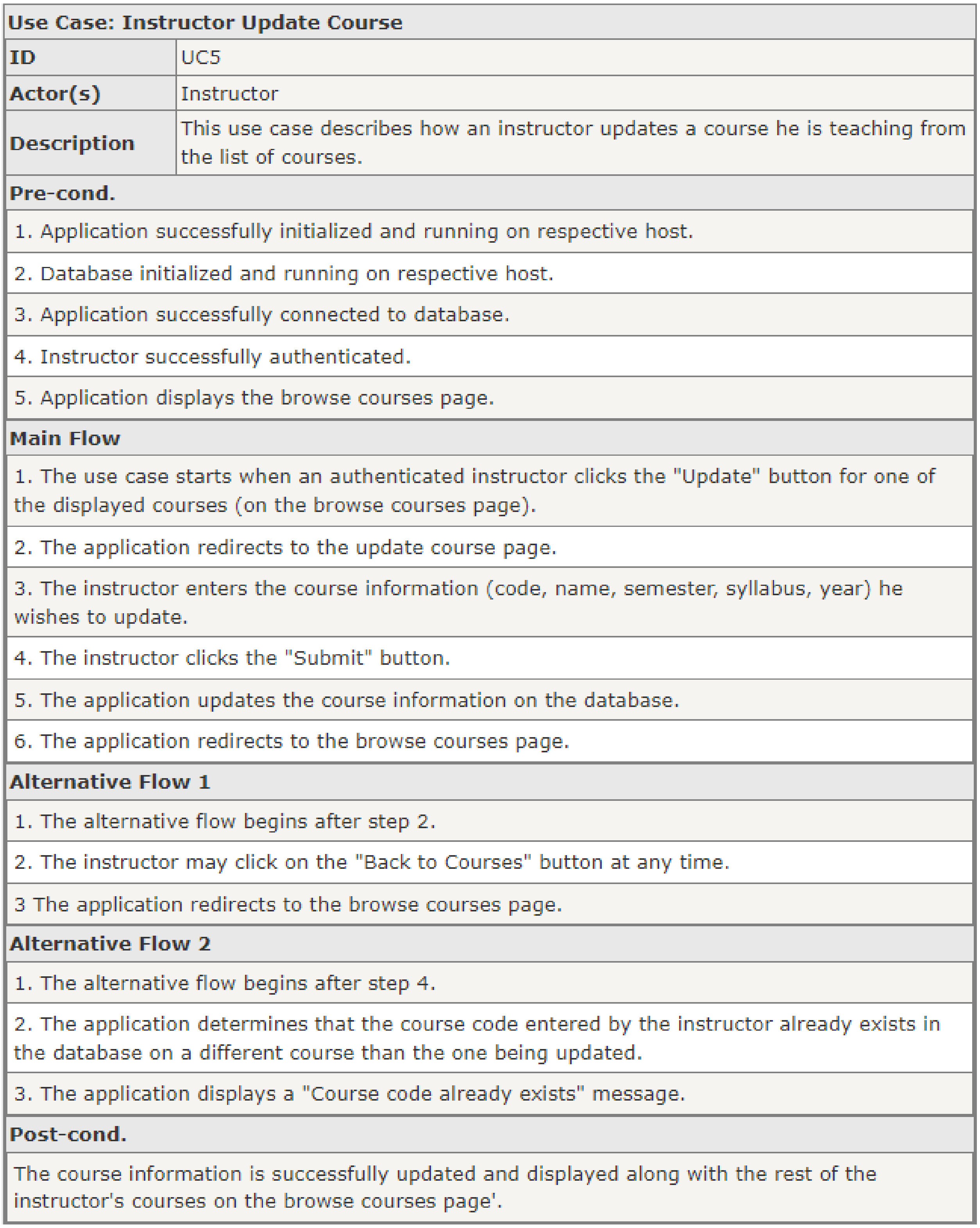


Use Case: Instructor Browse Courses	
ID	UC2
Actor(s)	Instructor
Description	This use case describes how an instructor browses the list of courses he is teaching.
Pre-cond.	
1. Application successfully initialized and running on respective host.	
2. Database initialized and running on respective host.	
3. Application successfully connected to database.	
4. Instructor successfully authenticated.	
Main Flow	
1. The use case starts when an authenticated instructor navigates or is redirected to the browse courses page.	
2. The application fetches the courses which correspond to the authenticated instructor from the database.	
3. The application displays the fetched courses on the courses page.	
Post-cond.	
-	

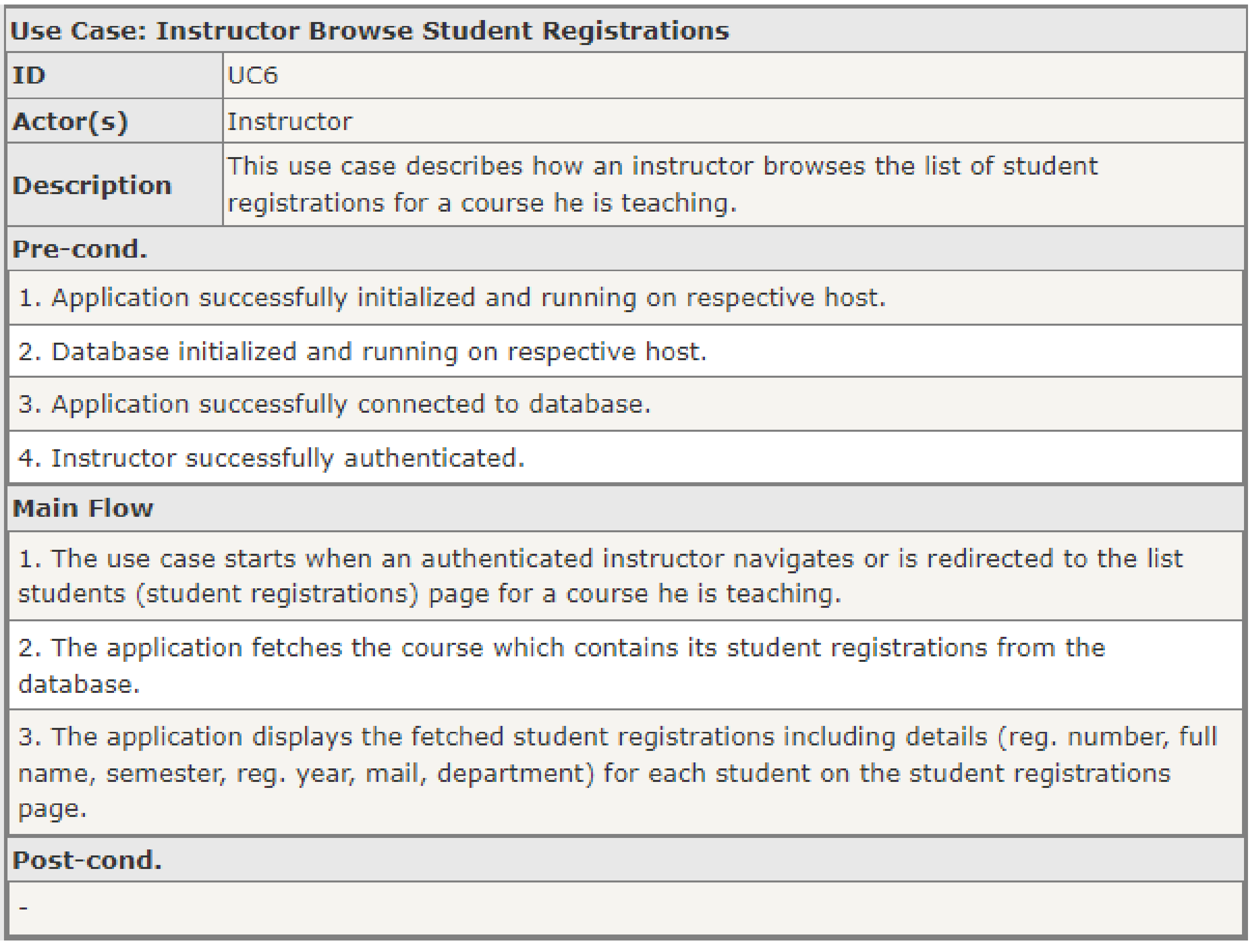
Use Case: Instructor Add Course	
ID	UC3
Actor(s)	Instructor
Description	This use case describes how an instructor adds a course he is teaching to the list of courses.
Pre-cond.	
1. Application successfully initialized and running on respective host.	
2. Database initialized and running on respective host.	
3. Application successfully connected to database.	
4. Instructor successfully authenticated.	
5. Application displays the browse courses page.	
Main Flow	
1. The use case starts when an authenticated instructor clicks the "Add Course" button (on the browse courses page).	
2. The application redirects to the add course page.	
3. The instructor enters the course information (code, name, semester, syllabus, year).	
4. The instructor clicks the "Submit" button.	
5. The application inserts the new course to the database.	
6. The application redirects to the browse courses page.	
Alternative Flow 1	
1. The alternative flow begins after step 2.	
2. The instructor may click on the "Back to Courses" button at any time.	
3 The application redirects to the browse courses page.	
Alternative Flow 2	
1. The alternative flow begins after step 4.	
2. The application determines that the course code entered by the instructor already exists in the database.	
3. The application displays a "Course code already exists" message.	
Post-cond.	
The new course is successfully added and displayed along with the rest of the instructor's courses on the browse courses page'.	



Use Case: Instructor Delete Course	
ID	UC4
Actor(s)	Instructor
Description	This use case describes how an instructor deletes a course from the list of courses.
Pre-cond.	
1. Application successfully initialized and running on respective host.	
2. Database initialized and running on respective host.	
3. Application successfully connected to database.	
4. Instructor successfully authenticated.	
5. Application displays the browse courses page.	
Main Flow	
1. The use case starts when an authenticated instructor clicks the "Delete" button for one of the displayed courses (on the browse courses page).	
2. The application displays a warning prompt asking "Are you sure you want to delete this course?".	
2.1 If the instructor clicks "OK" on the warning prompt.	
2.2 The application deletes the course from the database.	
Alternative Flow	
1. The alternative flow begins after step 2.	
2.1 If the instructor clicks "Cancel" on the warning prompt.	
2.2 The application does not delete the course from the database.	
Post-cond.	
The course is successfully deleted from the database and is no longer displayed along with the rest of the instructor's courses on the browse courses page'.	



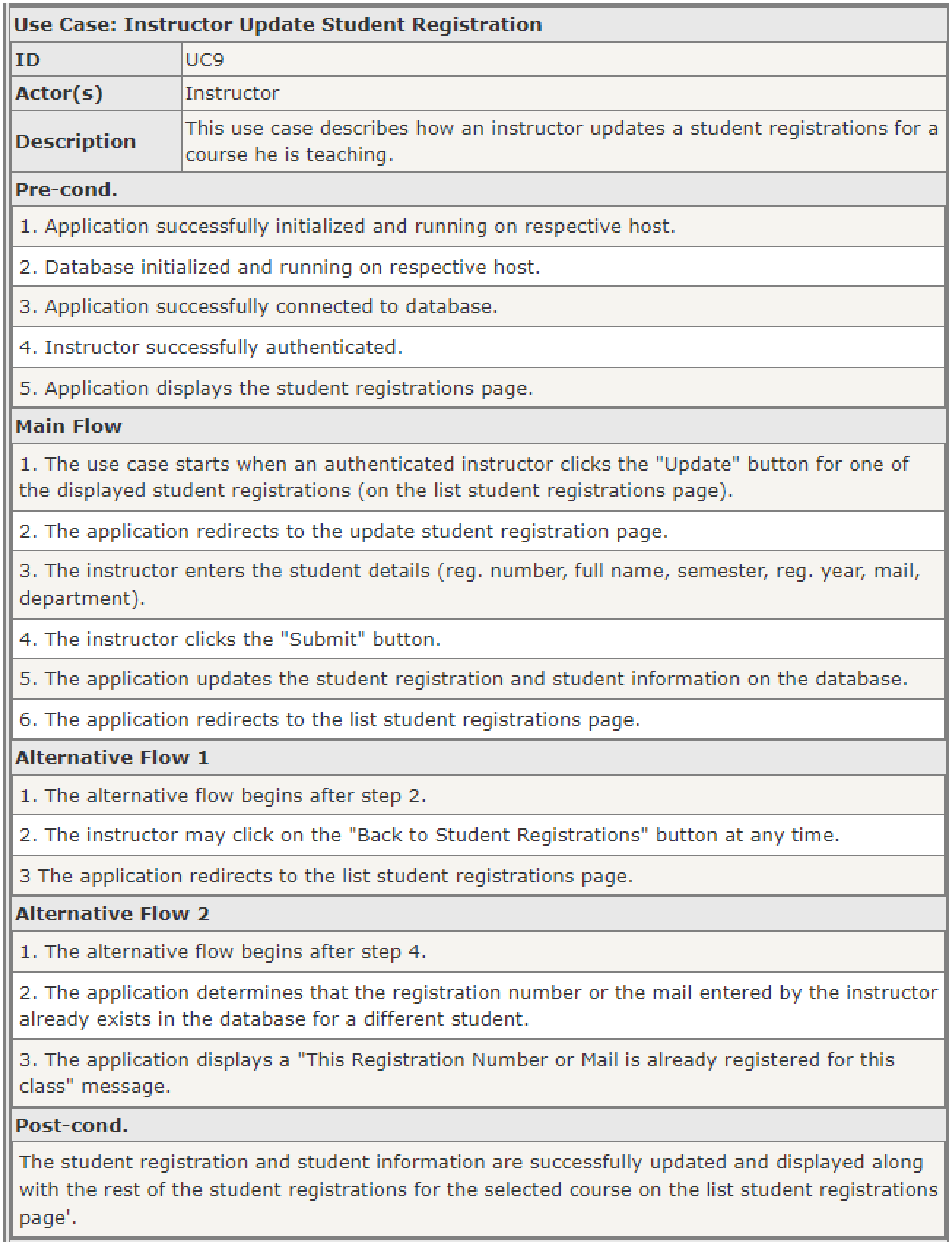
Use Case: Instructor Update Course	
ID	UC5
Actor(s)	Instructor
Description	This use case describes how an instructor updates a course he is teaching from the list of courses.
Pre-cond.	
1. Application successfully initialized and running on respective host.	
2. Database initialized and running on respective host.	
3. Application successfully connected to database.	
4. Instructor successfully authenticated.	
5. Application displays the browse courses page.	
Main Flow	
1. The use case starts when an authenticated instructor clicks the "Update" button for one of the displayed courses (on the browse courses page).	
2. The application redirects to the update course page.	
3. The instructor enters the course information (code, name, semester, syllabus, year) he wishes to update.	
4. The instructor clicks the "Submit" button.	
5. The application updates the course information on the database.	
6. The application redirects to the browse courses page.	
Alternative Flow 1	
1. The alternative flow begins after step 2.	
2. The instructor may click on the "Back to Courses" button at any time.	
3 The application redirects to the browse courses page.	
Alternative Flow 2	
1. The alternative flow begins after step 4.	
2. The application determines that the course code entered by the instructor already exists in the database on a different course than the one being updated.	
3. The application displays a "Course code already exists" message.	
Post-cond.	
The course information is successfully updated and displayed along with the rest of the instructor's courses on the browse courses page'.	



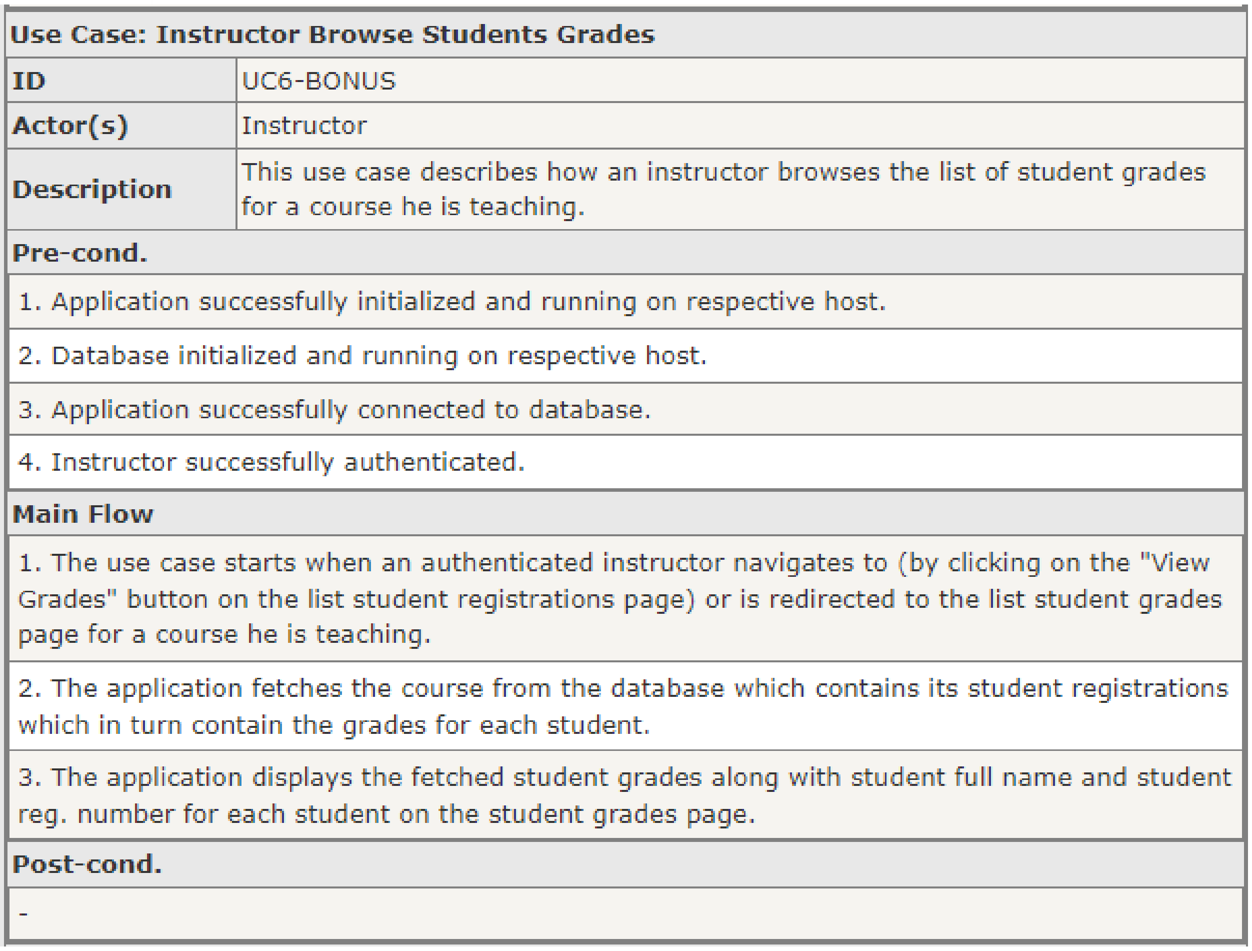
Use Case: Instructor Browse Student Registrations	
ID	UC6
Actor(s)	Instructor
Description	This use case describes how an instructor browses the list of student registrations for a course he is teaching.
Pre-cond.	
1. Application successfully initialized and running on respective host.	
2. Database initialized and running on respective host.	
3. Application successfully connected to database.	
4. Instructor successfully authenticated.	
Main Flow	
1. The use case starts when an authenticated instructor navigates or is redirected to the list students (student registrations) page for a course he is teaching.	
2. The application fetches the course which contains its student registrations from the database.	
3. The application displays the fetched student registrations including details (reg. number, full name, semester, reg. year, mail, department) for each student on the student registrations page.	
Post-cond.	
-	

Use Case: Instructor Add Student Registration	
ID	UC7
Actor(s)	Instructor
Description	This use case describes how an instructor adds a student registration for a course he is teaching.
Pre-cond.	
1. Application successfully initialized and running on respective host.	
2. Database initialized and running on respective host.	
3. Application successfully connected to database.	
4. Instructor successfully authenticated.	
5. Application displays the student registrations page.	
Main Flow	
1. The use case starts when an authenticated instructor clicks the "Add Student" button (on the list student registrations page).	
2. The application redirects to the add student registration page.	
3. The instructor enters the student details (reg. number, full name, semester, reg. year, mail, department).	
4. The instructor clicks the "Submit" button.	
5. The application inserts the new student registration to the database.	
5.1 If the student does not exist on the respective database table.	
5.2 The application inserts the new student on the respective database table.	
6. The application redirects to the browse list student registrations page.	
Alternative Flow 1	
1. The alternative flow begins after step 2.	
2. The instructor may click on the "Back to Student Registrations" button at any time.	
3 The application redirects to the list student registrations page.	
Alternative Flow 2	
1. The alternative flow begins after step 4.	
2. The application determines that the registration number or the mail entered by the instructor already exists in the database for a different student.	
3. The application displays a "This Registration Number or Mail is already registered for this class" message.	
Post-cond.	
The new student registration and student details are successfully added and displayed along with the rest of the student registrations for the selected course on the list student registrations page'.	

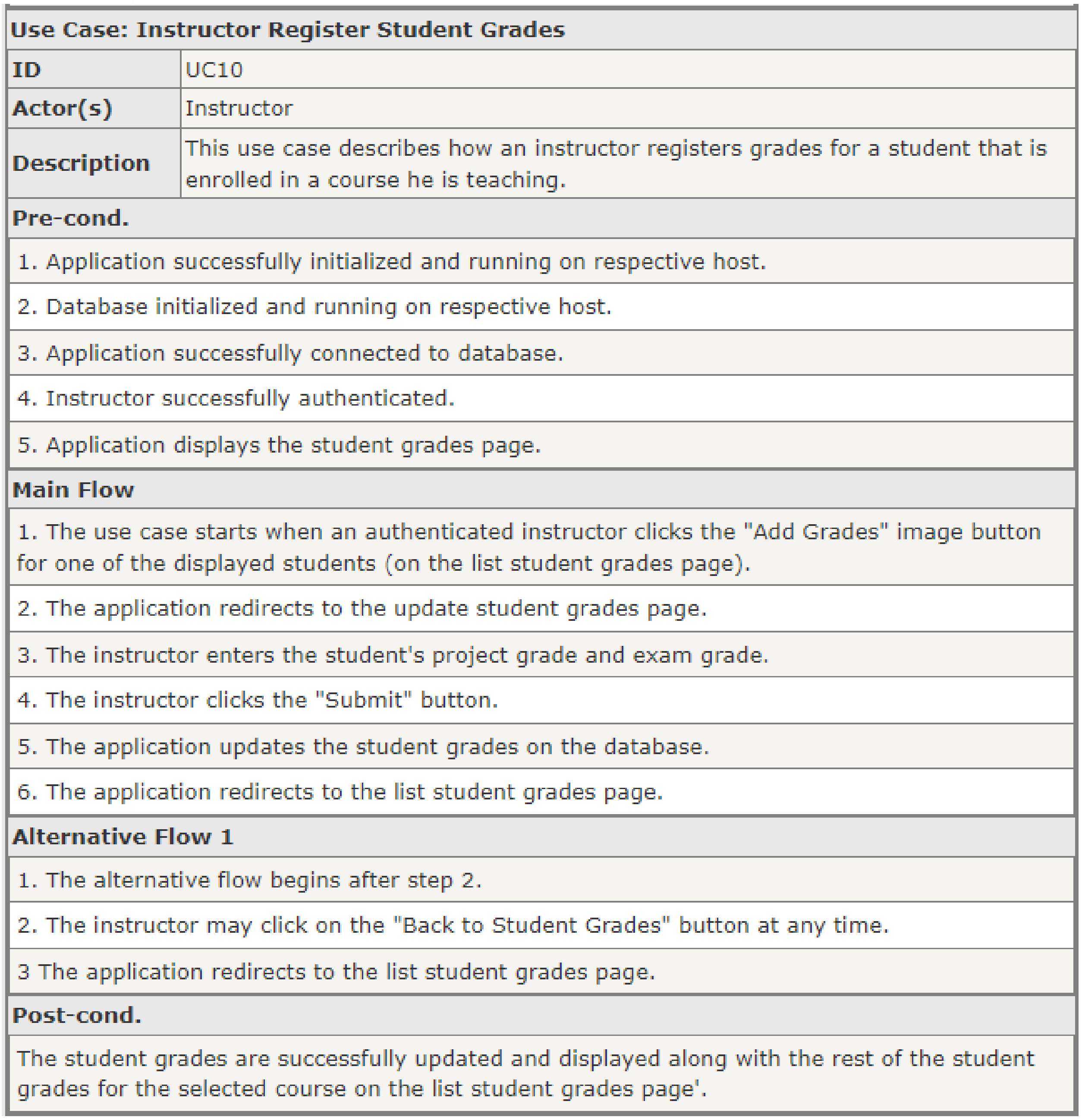
Use Case: Instructor Delete Student Registration	
ID	UC8
Actor(s)	Instructor
Description	This use case describes how an instructor deletes a student registration for a course he is teaching.
Pre-cond.	
1. Application successfully initialized and running on respective host.	
2. Database initialized and running on respective host.	
3. Application successfully connected to database.	
4. Instructor successfully authenticated.	
5. Application displays the student registrations page.	
Main Flow	
1. The use case starts when an authenticated instructor clicks the "Delete" button for one of the displayed student registrations (on the list student registrations page).	
2. The application displays a warning prompt asking "Are you sure you want to delete this student?".	
2.1 If the instructor clicks "OK" on the warning prompt.	
2.2 The application deletes the student registration from the database.	
2.2.1 If the selected student is not enrolled in any other course.	
2.2.2 The application deletes the student from the database.	
Alternative Flow	
1. The alternative flow begins after step 2.	
2.1 If the instructor clicks "Cancel" on the warning prompt.	
2.2 The application does not delete the student registration (or the student) from the database.	
Post-cond.	
The student registration (and possibly student) is successfully deleted from the database and is no longer displayed along with the rest of the student registrations for the selected course on the list student registrations page'.	



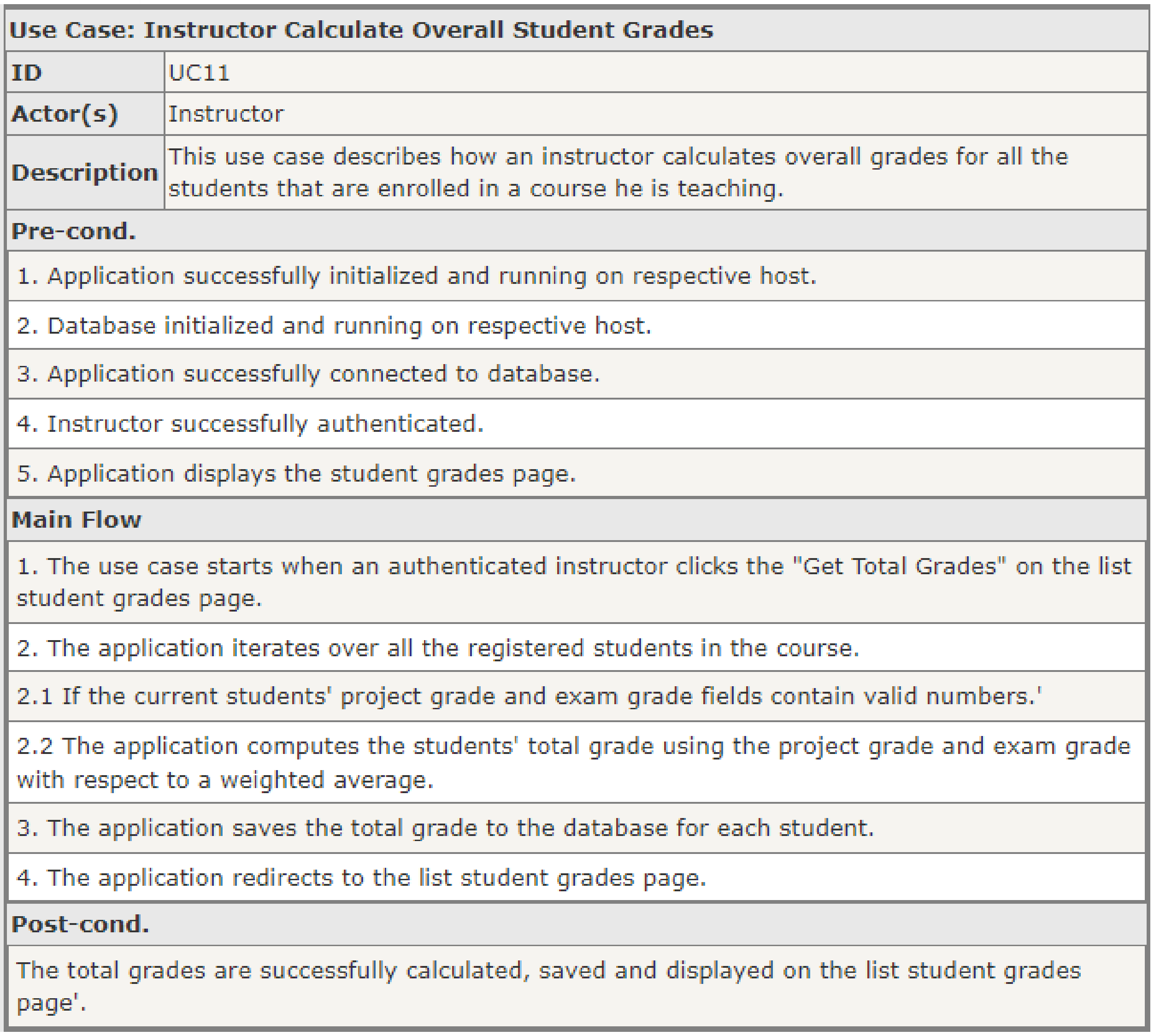
Use Case: Instructor Update Student Registration	
ID	UC9
Actor(s)	Instructor
Description	This use case describes how an instructor updates a student registrations for a course he is teaching.
Pre-cond.	
1. Application successfully initialized and running on respective host.	
2. Database initialized and running on respective host.	
3. Application successfully connected to database.	
4. Instructor successfully authenticated.	
5. Application displays the student registrations page.	
Main Flow	
1. The use case starts when an authenticated instructor clicks the "Update" button for one of the displayed student registrations (on the list student registrations page).	
2. The application redirects to the update student registration page.	
3. The instructor enters the student details (reg. number, full name, semester, reg. year, mail, department).	
4. The instructor clicks the "Submit" button.	
5. The application updates the student registration and student information on the database.	
6. The application redirects to the list student registrations page.	
Alternative Flow 1	
1. The alternative flow begins after step 2.	
2. The instructor may click on the "Back to Student Registrations" button at any time.	
3 The application redirects to the list student registrations page.	
Alternative Flow 2	
1. The alternative flow begins after step 4.	
2. The application determines that the registration number or the mail entered by the instructor already exists in the database for a different student.	
3. The application displays a "This Registration Number or Mail is already registered for this class" message.	
Post-cond.	
The student registration and student information are successfully updated and displayed along with the rest of the student registrations for the selected course on the list student registrations page'.	



Use Case: Instructor Browse Students Grades	
ID	UC6-BONUS
Actor(s)	Instructor
Description	This use case describes how an instructor browses the list of student grades for a course he is teaching.
Pre-cond.	
1. Application successfully initialized and running on respective host.	
2. Database initialized and running on respective host.	
3. Application successfully connected to database.	
4. Instructor successfully authenticated.	
Main Flow	
1. The use case starts when an authenticated instructor navigates to (by clicking on the "View Grades" button on the list student registrations page) or is redirected to the list student grades page for a course he is teaching.	
2. The application fetches the course from the database which contains its student registrations which in turn contain the grades for each student.	
3. The application displays the fetched student grades along with student full name and student reg. number for each student on the student grades page.	
Post-cond.	
-	



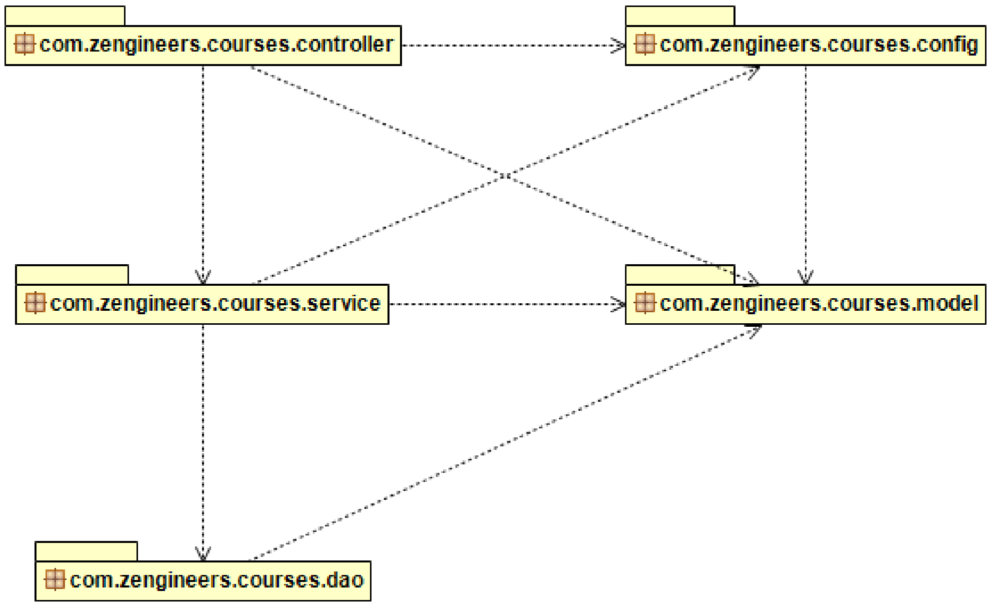
Use Case: Instructor Register Student Grades	
ID	UC10
Actor(s)	Instructor
Description	This use case describes how an instructor registers grades for a student that is enrolled in a course he is teaching.
Pre-cond.	
1. Application successfully initialized and running on respective host.	
2. Database initialized and running on respective host.	
3. Application successfully connected to database.	
4. Instructor successfully authenticated.	
5. Application displays the student grades page.	
Main Flow	
1. The use case starts when an authenticated instructor clicks the "Add Grades" image button for one of the displayed students (on the list student grades page).	
2. The application redirects to the update student grades page.	
3. The instructor enters the student's project grade and exam grade.	
4. The instructor clicks the "Submit" button.	
5. The application updates the student grades on the database.	
6. The application redirects to the list student grades page.	
Alternative Flow 1	
1. The alternative flow begins after step 2.	
2. The instructor may click on the "Back to Student Grades" button at any time.	
3 The application redirects to the list student grades page.	
Post-cond.	
The student grades are successfully updated and displayed along with the rest of the student grades for the selected course on the list student grades page'.	



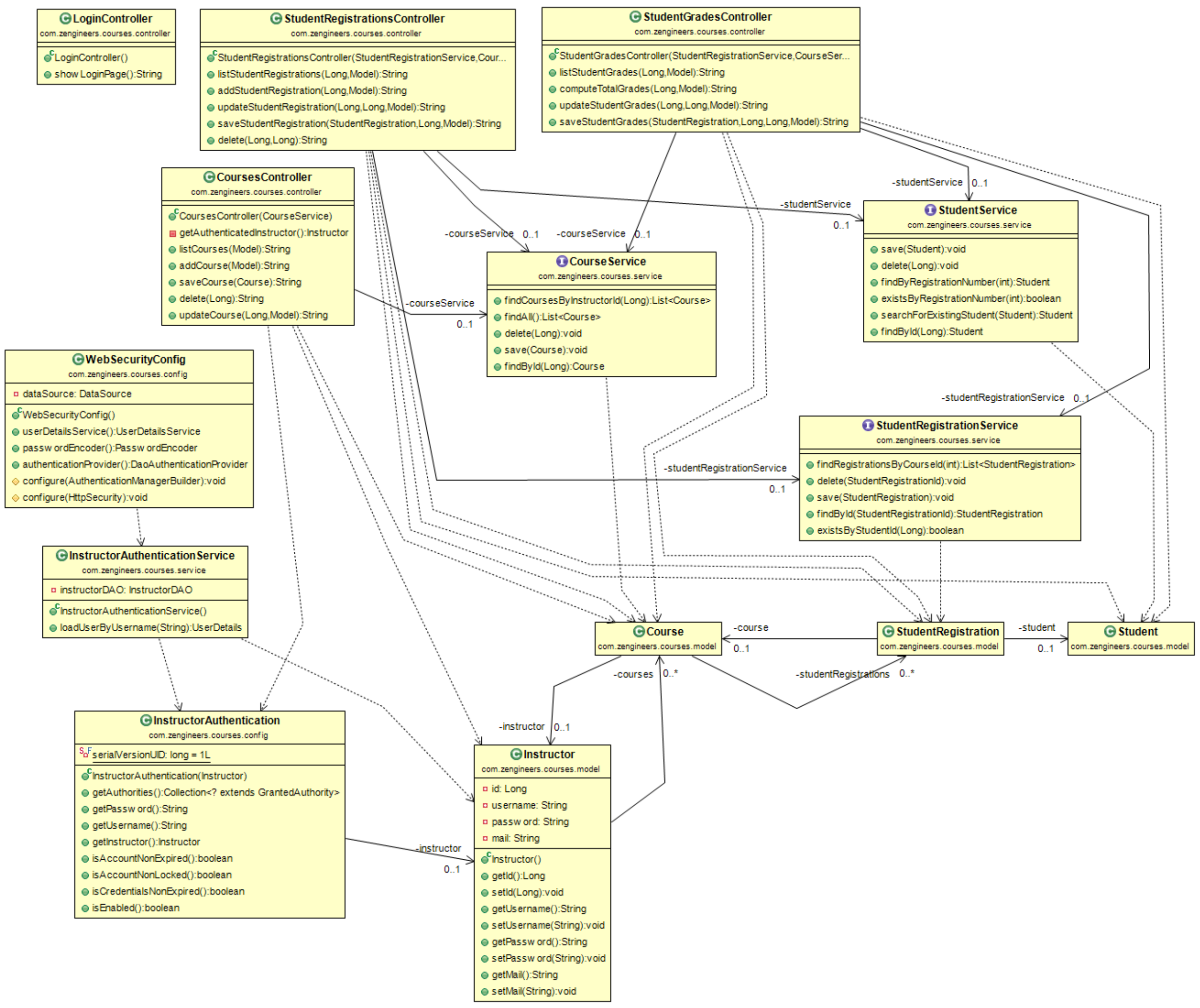
Use Case: Instructor Calculate Overall Student Grades	
ID	UC11
Actor(s)	Instructor
Description	This use case describes how an instructor calculates overall grades for all the students that are enrolled in a course he is teaching.
Pre-cond.	
1. Application successfully initialized and running on respective host.	
2. Database initialized and running on respective host.	
3. Application successfully connected to database.	
4. Instructor successfully authenticated.	
5. Application displays the student grades page.	
Main Flow	
1. The use case starts when an authenticated instructor clicks the "Get Total Grades" on the list student grades page.	
2. The application iterates over all the registered students in the course.	
2.1 If the current students' project grade and exam grade fields contain valid numbers.'	
2.2 The application computes the students' total grade using the project grade and exam grade with respect to a weighted average.	
3. The application saves the total grade to the database for each student.	
4. The application redirects to the list student grades page.	
Post-cond.	
The total grades are successfully calculated, saved and displayed on the list student grades page'.	

ARCHITECTURE & DESIGN

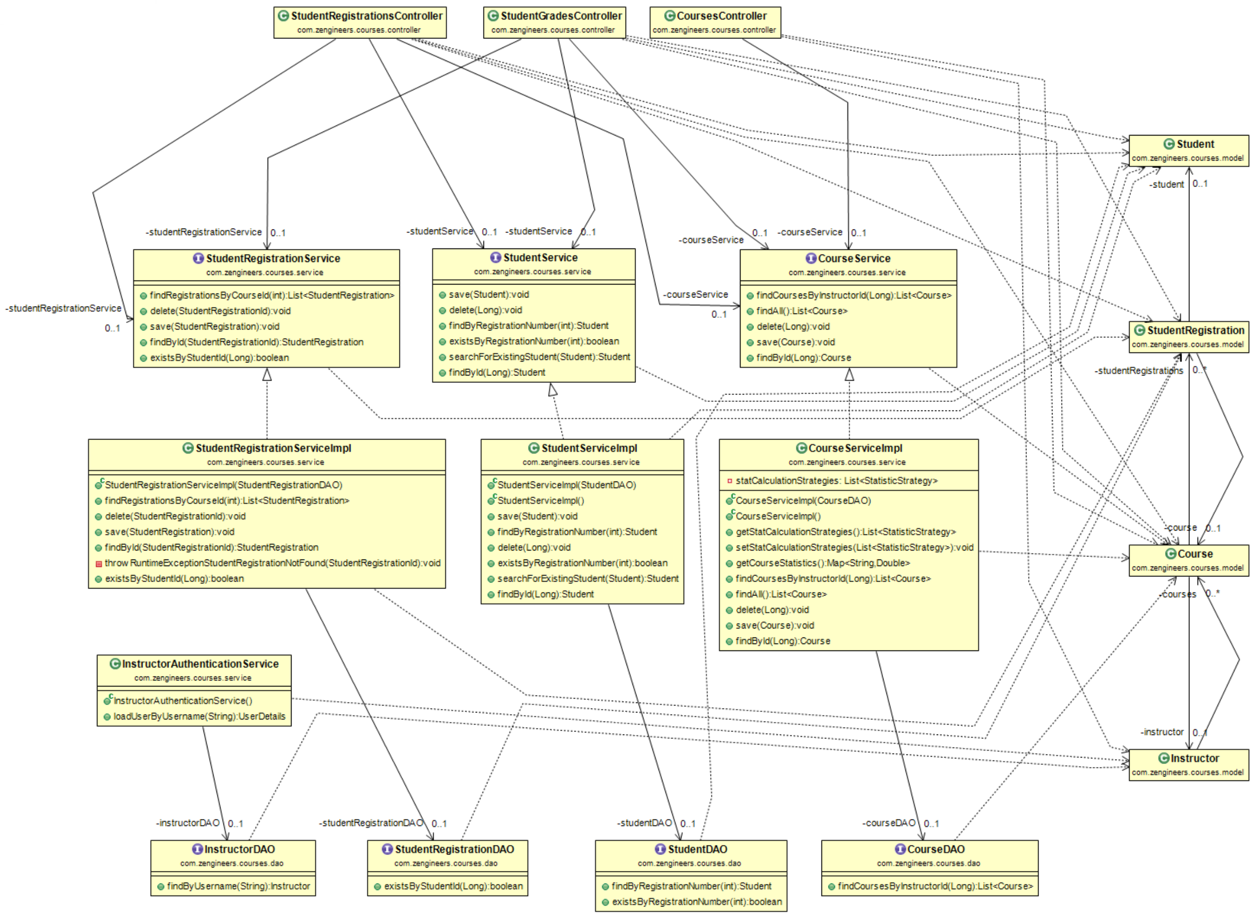
The overall application architecture is outlined in the following UML package diagram:


com.zengineers.courses.controller
com.zengineers.courses.config
com.zengineers.courses.service
com.zengineers.courses.model
com.zengineers.courses.dao

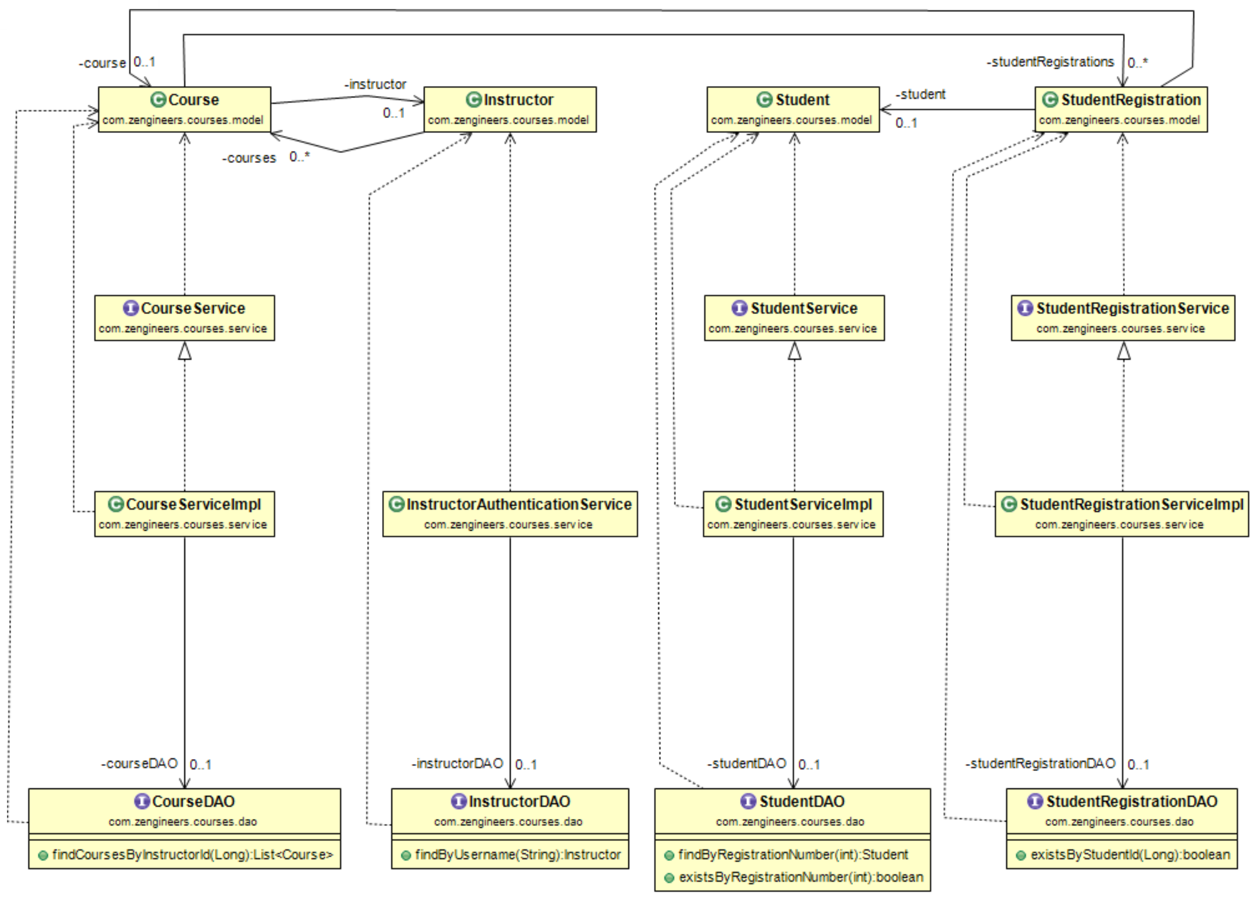
The detailed design is outlined by the following UML class diagrams.


LoginController
com.zengineers.courses.controller
LoginController()
showLoginPage():String
StudentRegistrationsController
com.zengineers.courses.controller
StudentRegistrationsController(StudentRegistrationService,Cour...
listStudentRegistrations(Long,Model):String
addStudentRegistration(Long,Model):String
updateStudentRegistration(Long,Long,Model):String
saveStudentRegistration(StudentRegistration,Long,Model):String
delete(Long,Long):String
StudentGradesController
com.zengineers.courses.controller
StudentGradesController(StudentRegistrationService,CourseSer...
listStudentGrades(Long,Model):String
computeTotalGrades(Long,Model):String
updateStudentGrades(Long,Long,Model):String
saveStudentGrades(StudentRegistration,Long,Long,Model):String
CoursesController
com.zengineers.courses.controller
CoursesController(CourseService)
getAuthenticatedInstructor():Instructor
listCourses(Model):String
addCourse(Model):String
saveCourse(Course):String
delete(Long):String
updateCourse(Long,Model):String
CourseService
com.zengineers.courses.service
findCoursesByInstructorId(Long):List<Course>
findAll():List<Course>
delete(Long):void
save(Course):void
findById(Long):Course
StudentService
com.zengineers.courses.service
save(Student):void
delete(Long):void
findByRegistrationNumber(int):Student
existsByRegistrationNumber(int):boolean
searchForExistingStudent(Student):Student
findById(Long):Student
WebSecurityConfig
com.zengineers.courses.config
dataSource: DataSource
WebSecurityConfig()
userDetailsService():UserDetailsService
passwordEncoder():PasswordEncoder
authenticationProvider():DaoAuthenticationProvider
configure(AuthenticationManagerBuilder):void
configure(HttpSecurity):void
StudentRegistrationService
com.zengineers.courses.service
findRegistrationsByCourseId(int):List<StudentRegistration>
delete(StudentRegistrationId):void
save(StudentRegistration):void
findById(StudentRegistrationId):StudentRegistration
existsByStudentId(Long):boolean
InstructorAuthenticationService
com.zengineers.courses.service
instructorDAO: InstructorDAO
InstructorAuthenticationService()
loadUserByUsername(String):UserDetails
Course
com.zengineers.courses.model
StudentRegistration
com.zengineers.courses.model
Student
com.zengineers.courses.model
InstructorAuthentication
com.zengineers.courses.config
serialVersionUID: long = 1L
InstructorAuthentication(Instructor)
getAuthorities():Collection&lt;? extends GrantedAuthority&gt;
getPassword():String
getUsername():String
getInstructor():Instructor
isAccountNonExpired():boolean
isAccountNonLocked():boolean
isCredentialsNonExpired():boolean
isEnabled():boolean
Instructor
com.zengineers.courses.model
id: Long
username: String
password: String
mail: String
Instructor()
getId():Long
setId(Long):void
getUsername():String
setUsername(String):void
getPassword():String
setPassword(String):void
getMail():String
setMail(String):void
-studentService 0..1
-studentService 0..1
-courseService 0..1
-courseService 0..1
-courseService 0..1
-studentRegistrationService 0..1
-studentRegistrationService 0..1
-course 0..1
-student 0..1
-courses 0..*
-studentRegistrations 0..*
-instructor 0..1
-instructor 0..1

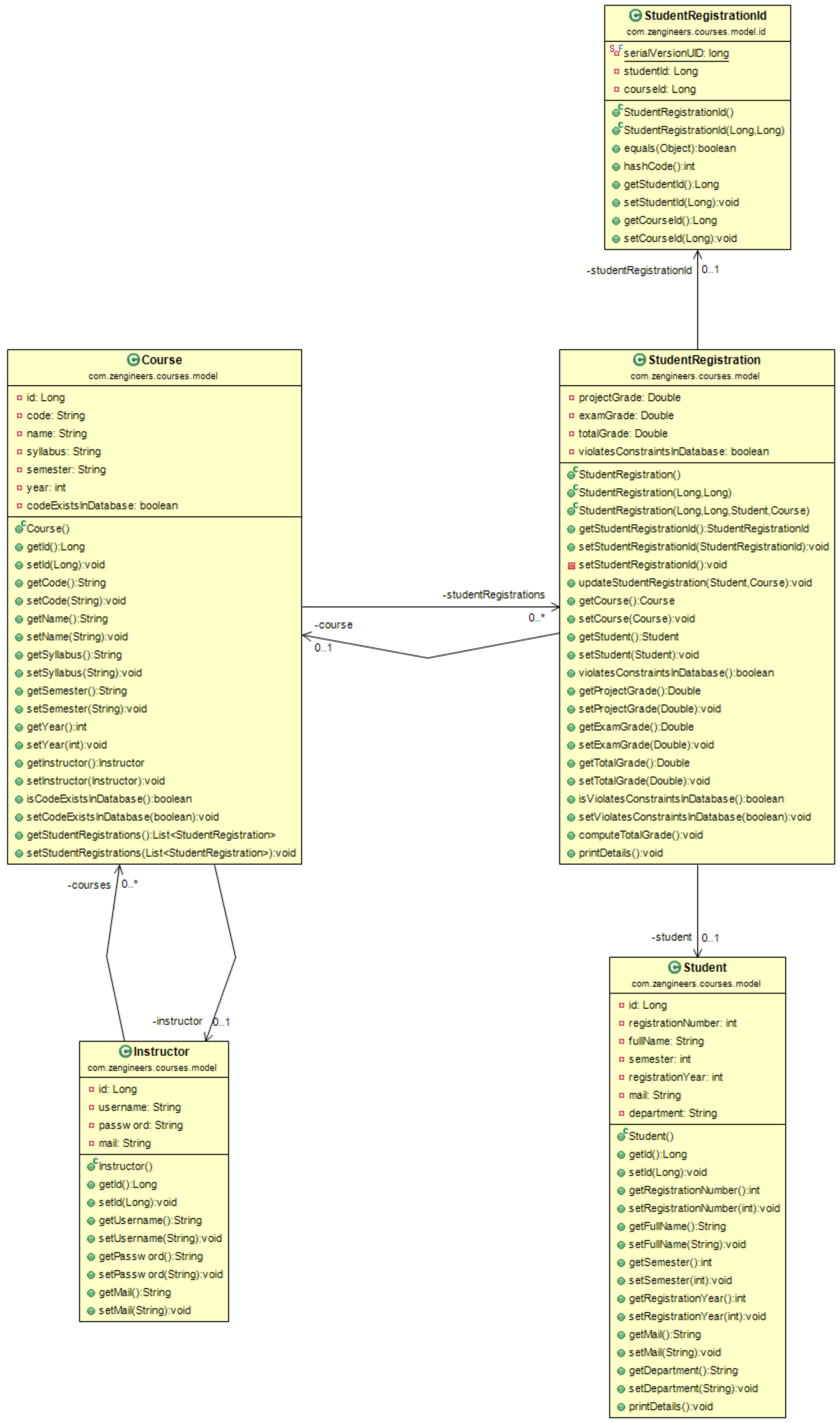
UML class diagram for the classes of the controller package and their associations


StudentRegistrationsController
com.zengineers.courses.controller
StudentGradesController
com.zengineers.courses.controller
CoursesController
com.zengineers.courses.controller
-studentRegistrationService 0..1
-studentService 0..1
-studentService 0..1
-courseService 0..1
-courseService 0..1
-studentRegistrationService 0..1
-courseService 0..1
StudentRegistrationService
com.zengineers.courses.service
findRegistrationsByCourseId(int):List<StudentRegistration>
delete(StudentRegistrationId):void
save(StudentRegistration):void
findById(StudentRegistrationId):StudentRegistration
existsByStudentId(Long):boolean
StudentService
com.zengineers.courses.service
save(Student):void
delete(Long):void
findByRegistrationNumber(int):Student
existsByRegistrationNumber(int):boolean
searchForExistingStudent(Student):Student
findById(Long):Student
CourseService
com.zengineers.courses.service
findCoursesByInstructorId(Long):List<Course>
findAll():List<Course>
delete(Long):void
save(Course):void
findById(Long):Course
StudentRegistrationServiceImpl
com.zengineers.courses.service
StudentRegistrationServiceImpl(StudentRegistrationDAO)
findRegistrationsByCourseId(int):List<StudentRegistration>
delete(StudentRegistrationId):void
save(StudentRegistration):void
findById(StudentRegistrationId):StudentRegistration
throw RuntimeExceptionStudentRegistrationNotFound(StudentRegistrationId):void
existsByStudentId(Long):boolean
StudentServiceImpl
com.zengineers.courses.service
StudentServiceImpl(StudentDAO)
StudentServiceImpl()
save(Student):void
findByRegistrationNumber(int):Student
delete(Long):void
existsByRegistrationNumber(int):boolean
searchForExistingStudent(Student):Student
findById(Long):Student
CourseServiceImpl
com.zengineers.courses.service
statCalculationStrategies: List<StatisticStrategy>
CourseServiceImpl(CourseDAO)
CourseServiceImpl()
getStatCalculationStrategies():List<StatisticStrategy>
setStatCalculationStrategies(List<StatisticStrategy>):void
getCourseStatistics():Map<String,Double>
findCoursesByInstructorId(Long):List<Course>
findAll():List<Course>
delete(Long):void
save(Course):void
findById(Long):Course
InstructorAuthenticationService
com.zengineers.courses.service
InstructorAuthenticationService()
loadUserByUsername(String):UserDetails
Student
com.zengineers.courses.model
-student 0..1
StudentRegistration
com.zengineers.courses.model
-studentRegistrations 0..*
-course 0..1
Course
com.zengineers.courses.model
-courses 0..*
-instructor 0..1
Instructor
com.zengineers.courses.model
-instructorDAO 0..1
-studentRegistrationDAO 0..1
-studentDAO 0..1
-courseDAO 0..1
InstructorDAO
com.zengineers.courses.dao
findByUsername(String):Instructor
StudentRegistrationDAO
com.zengineers.courses.dao
existsByStudentId(Long):boolean
StudentDAO
com.zengineers.courses.dao
findByRegistrationNumber(int):Student
existsByRegistrationNumber(int):boolean
CourseDAO
com.zengineers.courses.dao
findCoursesByInstructorId(Long):List<Course>

UML class diagram for the classes of the service package and their associations


Course
com.zengineers.courses.model
Instructor
com.zengineers.courses.model
Student
com.zengineers.courses.model
StudentRegistration
com.zengineers.courses.model
-course 0..1
-instructor 0..1
-courses 0..*
-student 0..1
-studentRegistrations 0..*
CourseService
com.zengineers.courses.service
StudentService
com.zengineers.courses.service
StudentRegistrationService
com.zengineers.courses.service
CourseServiceImpl
com.zengineers.courses.service
InstructorAuthenticationService
com.zengineers.courses.service
StudentServiceImpl
com.zengineers.courses.service
StudentRegistrationServiceImpl
com.zengineers.courses.service
-courseDAO 0..1
-instructorDAO 0..1
-studentDAO 0..1
-studentRegistrationDAO 0..1
CourseDAO
com.zengineers.courses.dao
findCoursesByInstructorId(Long):List<Course>
InstructorDAO
com.zengineers.courses.dao
findByUsername(String):Instructor
StudentDAO
com.zengineers.courses.dao
findByRegistrationNumber(int):Student
existsByRegistrationNumber(int):boolean
StudentRegistrationDAO
com.zengineers.courses.dao
existsByStudentId(Long):boolean

UML class diagram for the classes of the dao package and their associations



UML class diagram for the classes of the model package

The detailed design is further documented by the following Class-Responsibilities-Collaborations Cards for all the classes included in this release:

CoursesController	
• Map http requests regarding the Course Domain Model to respective pages	• CourseService

LoginController	
• Map http requests regarding user authentication to respective pages	• -

StudentGradesController	
• Map http requests regarding the Student Grades field of the StudentRegistration DomainModel to respective pages	• StudentRegistrationService • CourseService • StudentService

StudentRegistrationsController	
• Map http requests regarding the StudentRegistration Domain Model to respective pages	• StudentRegistrationService • CourseService • StudentService

InstructorAuthentication	
• Grant authentication to an Instructor user	• Instructor

WebSecurityConfig	WebSecurityConfigurerAdapter
• Defines pages that require user authentication to access	• -

Interface **CourseDAO**	
• Maps Course Domain Objects to the MySQL courses table row data	• Course

Interface **InstructorDAO**	
• Maps Instructor Domain Objects to the MySQL instructors table row data	• Instructor

Interface **StudentDAO**	
• Maps Student Domain Objects to the MySQL students table row data	• Student

Interface **StudentRegistrationDAO**	
• Maps StudentRegistration Domain Objects to the MySQL student_registrations table row data	• StudentRegistration

Course	
• Defines the data representation for the MySQL courses table row data	• -

Instructor	
• Defines the data representation for the MySQL instructors table row data	• -

Student	
• Defines the data representation for the MySQL students table row data	• -

StudentRegistration	
• Defines the data representation for the MySQL student_registrations table row data	• -

StudentRegistrationId	
• Defines the data representation for the composite primary key of the MySQL student_registrations table row data	

Interface CourseService	
• Defines the operations of the services that are provided by the application to the users regarding the Course Domain Model	• -

CourseServiceImpl	
• Contains an implementation of the operations of the services that are provided by the application to the users regarding the Course Domain Model	• CourseDAO

InstructorAuthenticationService	
• Implements the operations of the services regarding authentication to an Instructor user	• InstructorDAO

Interface StudentRegistrationService	
• Defines the operations of the services that are provided by the application to the users regarding the StudentRegistration Domain Model	• -

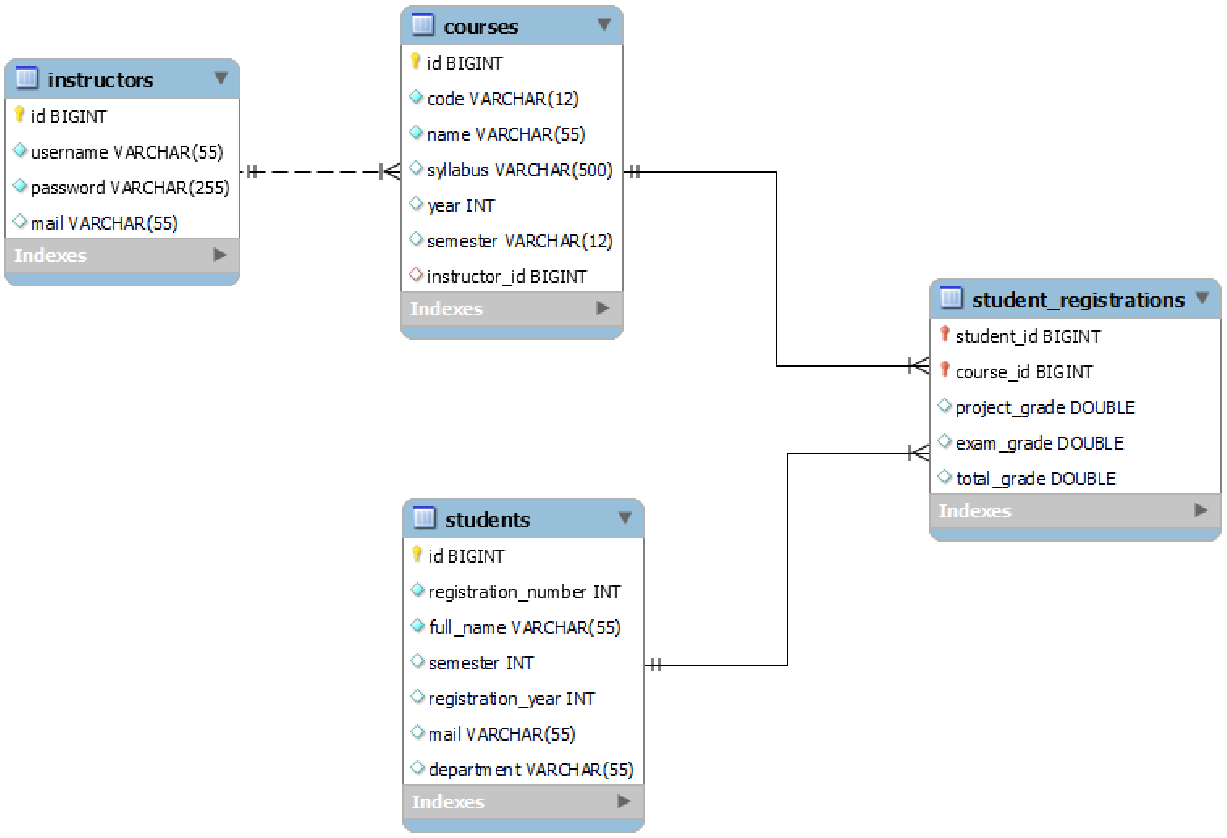
StudentRegistrationServiceImpl	
• Contains an implementation of the operations of the services that are provided by the application to the users regarding the StudentRegistration Domain Model	• StudentRegistrationDAO

Interface StudentService	
• Defines the operations of the services that are provided by the application to the users regarding the Student Domain Model	

StudentServiceImpl	
• Contains an implementation of the operations of the services that are provided by the application to the users regarding the Student Domain Model	• StudentDAO

THE DATABASE

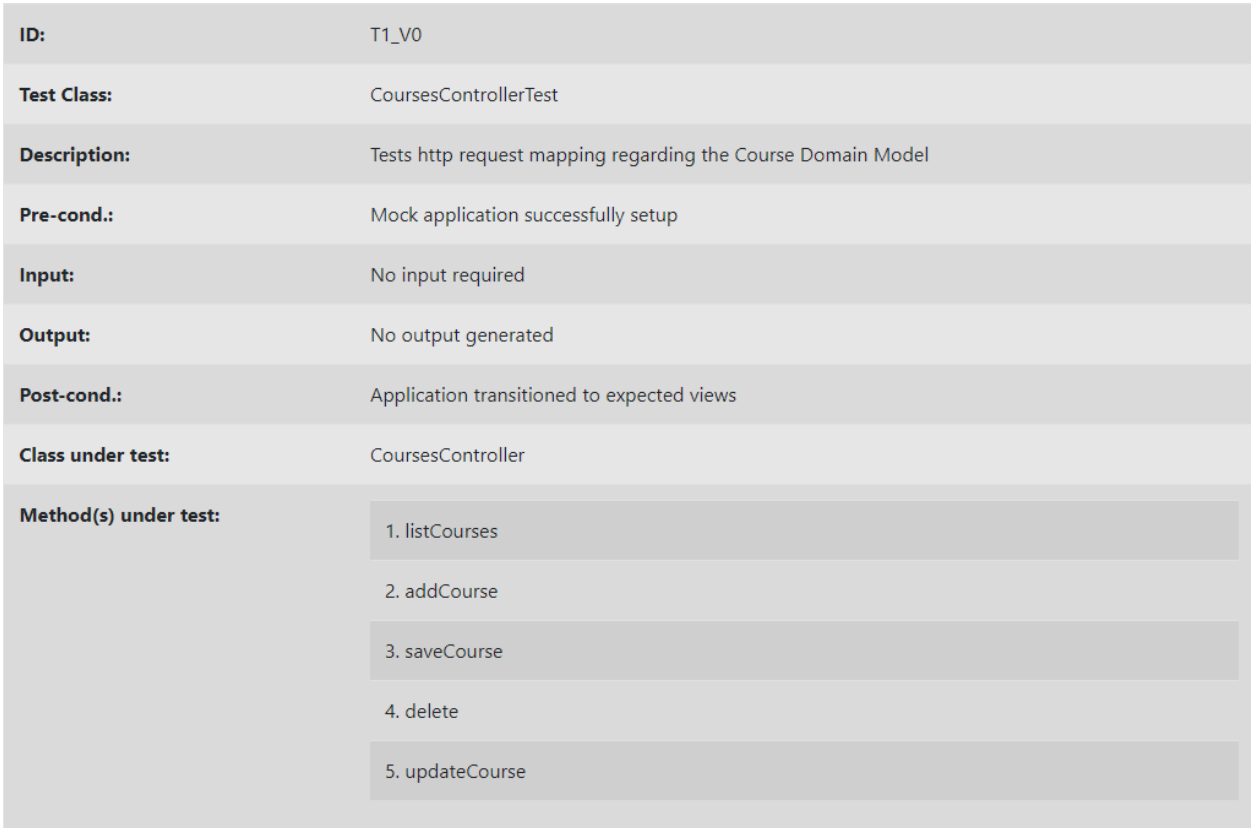
The database is illustrated by the following schema. Tables students and student_registrations were normalized so that data duplication is prevented if a student has multiple student registrations.



TEST CASES

Test cases have been developed to assert the functionality of the application's user stories. To assist with testing, the MockObjectGenerator class has been developed under the com.zengineers.courses test package. The purpose of which is to generate mock objects required by some of the tests. The test cases provide roughly a coverage of 90% of the source code.

The following test cases have been developed: (The test cases can also be found in json format in the respective project folder)



ID:	T1_V0
Test Class:	CoursesControllerTest
Description:	Tests http request mapping regarding the Course Domain Model
Pre-cond.:	Mock application successfully setup
Input:	No input required
Output:	No output generated
Post-cond.:	Application transitioned to expected views
Class under test:	CoursesController
Method(s) under test:	1. listCourses 2. addCourse 3. saveCourse 4. delete 5. updateCourse

ID:	T2_V0
Test Class:	LoginControllerTest
Description:	Tests http request mapping regarding user authentication
Pre-cond.:	Mock application successfully setup
Input:	No input required
Output:	No output generated
Post-cond.:	Application transitioned to expected view
Class under test:	LoginController
Method(s) under test:	1. showLoginPage

ID:	T3_V0
Test Class:	StudentGradesControllerTest
Description:	Tests http request mapping regarding student grades fields of the StudentRegistration Domain Model
Pre-cond.:	Mock application successfully setup
Input:	Mock StudentRegistration, including Student and Course objects and studentId and courseId
Output:	No output generated
Post-cond.:	Application transitioned to expected views
Class under test:	StudentGradesController
Method(s) under test:	1. listStudentGrades 2. computeTotalGrades 3. updateStudentGrades 4. saveStudentGrades

ID:	T4_V0
Test Class:	StudentRegistrationsControllerTest
Description:	Tests http request mapping regarding the StudentRegistration Domain Model
Pre-cond.:	Mock application successfully setup
Input:	Mock StudentRegistration, including Student and Course objects and studentId and courseId
Output:	No output generated
Post-cond.:	Application transitioned to expected views
Class under test:	StudentRegistrationsController
Method(s) under test:	1. listStudentRegistrations 2. addStudentRegistration 3. updateStudentRegistration 4. saveStudentRegistration 5. delete

ID:	T5_V0
Test Class:	CourseDAOTest
Description:	Tests MySQL courses table row data mapping to Course Domain Objects
Pre-cond.:	Database running
Input:	List of mock Course objects which correspond to the courses an instructor is teaching
Output:	No output generated
Post-cond.:	Application fetched correct data from the database
Class under test:	CourseDAO
Method(s) under test:	1. findCoursesByInstructorId

ID:	T6_V0
Test Class:	InstructorDAOTest
Description:	Tests MySQL instructors table row data mapping to Instructor Domain Objects
Pre-cond.:	Database running
Input:	Instructor information
Output:	No output generated
Post-cond.:	Application fetched correct data from the database
Class under test:	InstructorDAO
Method(s) under test:	1. findByUsername

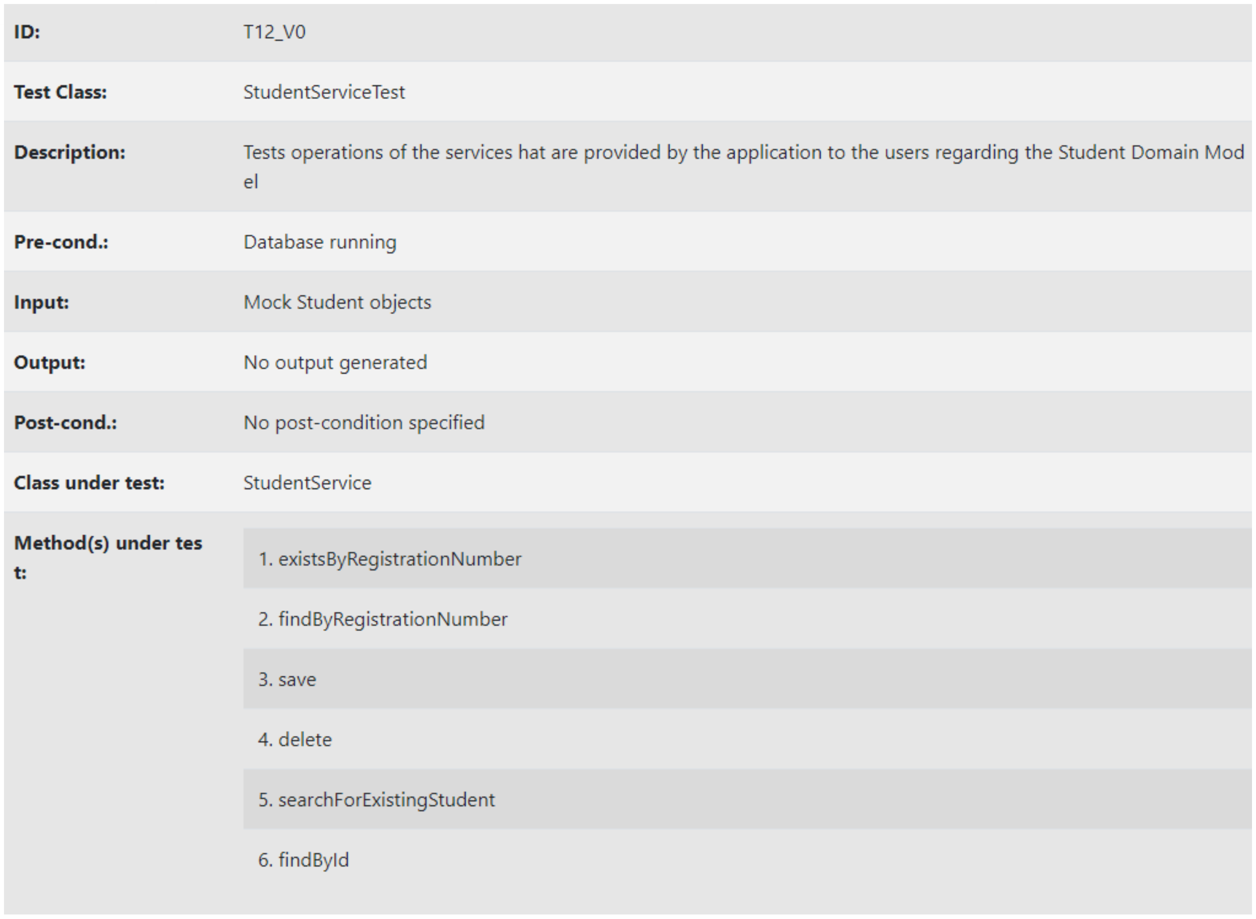
ID:	T7_V0
Test Class:	StudentDAOTest
Description:	Tests MySQL students table row data mapping to Student Domain Objects
Pre-cond.:	Database running
Input:	Mock Student object
Output:	No output generated
Post-cond.:	Application fetched correct data from the database
Class under test:	StudentDAO
Method(s) under test:	1. findByRegistrationNumber 2. existsByRegistrationNumber

ID:	T8_V0
Test Class:	StudentRegistrationDAOTest
Description:	Tests MySQL student_registrations table row data mapping to StudentRegistration Domain Objects
Pre-cond.:	Database running
Input:	StudentId
Output:	No output generated
Post-cond.:	Application fetched correct data from the database
Class under test:	StudentRegistrationDAO
Method(s) under test:	1. existsByStudentId

ID:	T9_V0
Test Class:	CourseServiceTest
Description:	Tests operations of the services that are provided by the application to the users regarding the Course Domain Model
Pre-cond.:	Database running
Input:	List of mock Course objects which correspond to the courses an instructor is teaching
Output:	No output generated
Post-cond.:	No post-condition specified
Class under test:	CourseService
Method(s) under test:	1. findAll 2. findCoursesByInstructorId 3. delete 4. save 5. findById

ID:	T10_V0
Test Class:	InstructorAuthenticationServiceTest
Description:	Tests operations of the services regarding authentication to an Instructor user
Pre-cond.:	Database running
Input:	Instructor username and (encrypted) password
Output:	No output generated
Post-cond.:	Instructor user successfuly authenticated or rejected due to bad credentials
Class under test:	InstructorAuthenticationService
Method(s) under test:	1. loadUserByUsername

ID:	T11_V0
Test Class:	StudentRegistrationServiceTest
Description:	Tests operations of the services hat are provided by the application to the users regarding the StudentRegistration Do main Model
Pre-cond.:	Database running
Input:	Mock StudentRegistration object which includes Student and Course objects
Output:	No output generated
Post-cond.:	No post-condition specified
Class under test:	StudentRegistrationService
Method(s) under te st:	1. findById 2. existsByStudentId 3. save 4. delete



ID:	T12_V0
Test Class:	StudentServiceTest
Description:	Tests operations of the services hat are provided by the application to the users regarding the Student Domain Mod el
Pre-cond.:	Database running
Input:	Mock Student objects
Output:	No output generated
Post-cond.:	No post-condition specified
Class under test:	StudentService
Method(s) under tes t:	1. existsByRegistrationNumber 2. findByRegistrationNumber 3. save 4. delete 5. searchForExistingStudent 6. findById