**Online Exam Management Software**

Final Report

**Slippery Rock University**

Brent Kosior bmk1019@sru.edu

Seth Chritzman sdc1004@sru.edu

Oleksii Dukhovenko oxd1005@sru.edu

Group Number: 4

Class: CPSC488 - 01

**Contents**

[**References 3**](#_snk3loz4ybmi)

[W3Schools. (n.d.). How to Create a Dropdown Navbar. https://www.w3schools.com/howto/howto\_css\_dropdown\_navbar.asp (Referenced in homepages for all the user types and style.css) 3](#_rjgrs8vwmrdt)

[References (continued) 4](#_ab4tu1ojzrkk)

[1. Completion 5](#_w25pc6glt5q3)

[2. Contributions 6](#_tizas5tvm1c2)

[**3. Problem Statement 7**](#_h7fa76pkr3pl)

[Background and Rationale 7](#_hlnqylqi2dsg)

[Problem Definition 7](#_eiu52vnjt55j)

[Project Goals 7](#_fhdzfzd5x4d7)

[Scope 8](#_6rf8zzdbmby6)

[**4. System Overview and Solutions 8**](#_lm9w4g6m8wbc)

[System Architecture 8](#_8lg5pff22xs7)

[Solution Strategy 11](#_sg8w9xjtv3jn)

[5. UML Diagrams 12](#_layaj88gqv5x)

[**6. System Requirements 12**](#_suqppurzp59q)

[Introduction 12](#_x1gl0u46og5a)

[System Development tools 12](#_fsvemp2h0r94)

[Hardware and Software Requirements 12](#_53ascu7k60yz)

[**7. Technical Specifications and Implementation 12**](#_qjaabgykt2ls)

[Technical Framework 12](#_vgtwm2xoqr2u)

[Implementation Process 13](#_sg2dk0whdtn4)

[Challenges and Resolutions 15](#_u47dpzetsx6t)

[**8. Testing and Validation 15**](#_l0sdllh390dn)

[9. Caveats/Minefields 22](#_acqmhx1038by)

[User Deletion and Editing 22](#_2t4py8tk2id3)

[Two Factor Authentication 22](#_kb9oawn7siq5)

[Login Page 22](#_qzzaumx4ttna)

[10. Required Classes 23](#_ejlu57enuorg)

[11. Code Reusability 27](#_oi0lfq8ls9gq)

[12. Deployment/Maintenance 28](#_idv7nuq4l77t)

[13. Post-Mortem Analysis 29](#_qj0tf0u3v9nx)

# 

# **References**

### W3Schools. (n.d.). How to Create a Dropdown Navbar.<https://www.w3schools.com/howto/howto_css_dropdown_navbar.asp> (Referenced in homepages for all the user types and style.css)

Maven Group ID: edu.sru.freilino.LoginPage Artifact ID: LoginPage (Referenced architectural ideas for a login page software design)

Spring Framework Guru. (2020, March 1). Spring Framework Tutorial. YouTube.<https://www.youtube.com/watch?v=xaR7Nk0E1Aw&t=5082s> (Video Title: Spring Framework Guru - Spring Framework Tutorial) (Video Publisher: Spring Framework Guru) (Date Published: March 1, 2020) (Date accessed: September 3, 2023)

Fadatare, R. (2023, July 11). Spring Boot User Registration and Login Example Tutorial. Java Guides.<https://www.javaguides.net/2018/10/user-registration-module-using-springboot-springmvc-springsecurity-hibernate5-thymeleaf-mysql.html>

GeeksforGeeks. (n.d.). Spring Boot JpaRepository with Example.<https://www.geeksforgeeks.org/spring-boot-jparepository-with-example/#> (Referenced in AdministratorRepository.java)

JavaTpoint. (n.d.). Spring MVC @RequestParam Annotation.<https://www.javatpoint.com/spring-mvc-requestparam-annotation> (Referenced in AdministratorController.java)

BezKoder. (n.d.). Spring Boot: Upload/Import Excel File Data into MySQL Database.<https://www.bezkoder.com/spring-boot-upload-excel-file-database/> (Referenced in "Export data to Excel from SpringBoot")

Spring.io. (n.d.). Getting Started | Accessing Data with MySQL.<https://spring.io/guides/gs/accessing-data-mysql/> (Referenced in "Accessing data with mySQL")

Baeldung. (n.d.). Many-To-Many Relationship in JPA.<https://www.baeldung.com/jpa-many-to-many> (Referenced in "Many to Many Associations")

Spring Security — Demonstrating Custom Authentication Success Handler. <https://mainul35.medium.com/spring-security-demonstrating-custom-authentication-success-handler-3b6fcb572a53> (Referenced in CustomAuthenticationSuccessHandler.java)

UserAuthentication | Redirection Based on roles:<https://www.codejava.net/frameworks/spring-boot/redirect-users-after-login-based-on-roles> (Referenced in CustomAuthenticationSuccessHandler.java)

### **References (continued)**

Redirect to Different Pages After Login With Spring Security. <https://www.baeldung.com/spring-redirect-after-login> (Referenced in CustomAuthenticationSuccessHandler.java)

ChatGPT, OpenAI. (2023). General project assistance and information provided for OnlineExamManagementSystem. No URL available. (General Reference).

Stack Overflow. (n.d.). Populating a MySQL database with values. Retrieved from <https://stackoverflow.com/questions/4475583/populating-a-mysql-database-with-values>

‌GeeksforGeeks. (n.d.). Different ways of Reading a text file in Java. GeeksforGeeks. Retrieved from https://www.geeksforgeeks.org/different-ways-reading-text-file-java/

Baeldung. (n.d.). How to Read a File in Java. Baeldung. Retrieved from <https://www.baeldung.com/reading-file-in-java>

Stack Overflow. (n.d.). Inserting records into a MySQL table using Java. Retrieved from <https://stackoverflow.com/questions/18270414/inserting-records-into-a-mysql-table-using-java>

(2023, October 21). *How Exam Generator software works?* [Video]. YouTube.<https://www.youtube.com/watch?v=uKU2N1vYctY>

Stack Overflow. (n.d.). How do I sort records in a text file using Java? Retrieved from<https://stackoverflow.com/questions/740936/how-do-i-sort-records-in-a-text-file-using-java>

Mockito. (n.d.). Mockito [Software]. Retrieved from <https://site.mockito.org/>

Byte Buddy. (n.d.). Byte Buddy (Version 1.14.9) [Software]. Retrieved from<https://github.com/raphw/byte-buddy>

### **1. Completion**

Completion Status: *Incomplete*

**Completed Modules**

* Login capabilities with Spring Boot security login
* Endpoint management based on user type
* Database integration and associations
* User management such as editing profiles and deletion of profiles
* Personal profile management and editing permissions
* Uploading entities via Excel and handling associations within
* Association between Entities
* Creating/Taking/Grading Exams
* Lower Level Domain(LLD) user management at Top Level Domain(TLD)
* Testing with regards to the J-Unit tests for main domain models

**Incomplete Modules**

The following are implementations that have not been fulfilled for the current version of OnlineExaminationManagementSoftware.

* **Not Tested**
  + Blackbox Testing
  + Whitebox Testing
* **Not Coded**
  + IP tracking for students taking exams.

### 

### 

### **2. Contributions**

**Group Contributions**

Our group has been actively engaged in the development of our project, contributing to a range of areas from frontend design to backend implementation. Collectively, our efforts have resulted in significant progress across the project's various components. The group's contribution to the project's codebase varied depending on the feature or functionality, with the total group contribution ranging between 90% to 100% for different segments of the project.

**External Resource Contributions**

To complement our work and ensure we adhered to industry standards, we integrated insights and code snippets from several reputable resources. These resources provided us with foundational knowledge and practical examples that were instrumental in shaping our project. The contribution from these resources is as follows:

YouTube: Used for a learning tool throughout the duration of this project, learning new methods, and designs.

W3Schools: 3% of the project code, for navigation bar design.

JavaCode, Medium: Was used throughout 5% throughout this project as a reference to Spring Security and Role Based Authorization

StackOverflow: 1% of the project code, exporting/importing via excel along with assisting in populating our database dynamically..

Maven Repository: 2% of the project code, for login page architecture.

Spring Framework Guru (YouTube): 6% of the project code, for Spring Framework tutorials.

Java Guides: 2% of the project code, for Spring Boot registration and login example.

GeeksforGeeks and JavaTpoint: Each contributed 4% of the project code for JpaRepository and parameter handling examples, respectively.

Bezkoder and Spring.io: Each contributed 1% to the project code for file handling and MySQL data access functionalities.

Baeldung: Two different articles from Baeldung contributed 2% and 3% respectively for understanding JPA relationships and custom authentication success handling.

ChatGTP OpenAI, was used generally throughout this project for helping us solve bugs and debug issues that arose throughout. (5%)

Overall, external resources contributed to the remaining 1% to 6% of the project code, depending on the specific area of development. This strategic integration of external expertise has greatly enhanced the quality and efficiency of our project outcomes. Altogether roughly adding up to 10% of our code coming from external sources.

In summary, our project is a testament to the synergy between the group's dedicated efforts and the rich pool of knowledge available from external technical resources. This collaborative approach has been pivotal in developing a comprehensive application that meets our project goals.

### 

### 

### **3. Problem Statement**

The Exam Management Software project, conducted by Seth, Brent, and Oleksii at CPSC group 4 at Slippery Rock University, aimed to address a critical need in the educational sector: the development of a robust, versatile, and secure online exam management system. This system was envisioned to cater to the diverse needs of educational institutions in conducting various types of exams, ranging from simple quizzes to complex assessments.

#### **Background and Rationale**

In the rapidly evolving educational landscape, where digital transformation is becoming increasingly significant, the reliance on traditional, paper-based examination methods has proven to be limiting. These methods are not only resource-intensive but also lack the flexibility needed to adapt to different teaching methodologies and assessment criteria. Furthermore, the onset of global challenges, such as the COVID-19 pandemic, has accelerated the need for reliable and efficient online exam management solutions.

#### **Problem Definition**

The primary challenge was to create a software system capable of handling the intricacies of exam management, including but not limited to, question generation, student management, exam scheduling, and results processing. Moreover, ensuring the integrity and security of the exams, especially in an online environment, was a paramount concern. The project sought to address these issues by developing a software solution that is not only efficient and user-friendly but also secure and adaptable to various academic requirements.

#### **Project Goals**

The key goals of the project were:

* To develop a user-friendly interface for both instructors and students, facilitating ease of use and accessibility.
* To implement a flexible question generation mechanism that can accommodate both manual and AI-driven approaches.
* To ensure the highest standards of security to prevent unauthorized access and maintain the integrity of the examination process.
* To create a scalable system that can be easily adapted or expanded to meet the changing needs of educational institutions.

#### **Scope**

The scope of the project encompassed the development of a comprehensive system that includes modules for exam creation, management, execution, and review. This system was designed to be compatible with various academic structures and examination formats, providing a one-stop solution for all exam-related activities in an educational setting.

### 

### **4. System Overview and Solutions**

#### **System Architecture**

The Exam Management Software is a multi-faceted system designed to streamline the exam management process in educational institutions. Its architecture is composed of several key components:

1. **Master Server**: Serves as the central hub for the entire system, managing database interactions, user authentication, and coordination between different modules. It ensures data integrity and security across the system.
2. **Administrator Interface**: This user type is enabled to view data such as a list of students that are enrolled in the system, along with instructors and Schedule managers. They have the authority to create schedule manager accounts and manage them.
3. **Schedule Manager Interface**: This user type is enabled to view all the instructors and students that are enrolled in the system, they are able to create and delete all these users. They are able to create courses and associate the students and instructors to courses.
4. **Instructor Interface**: This interface that allows instructors to create, manage, and review exams. Instructors can upload questions manually or use the AI-driven option for automated question generation. This interface also includes tools for setting exam parameters like duration, format, and grading criteria.
5. **Student Interface**: This interface allows students to take exams, view scheduled tests, and access their results. It features a responsive design to accommodate various devices and screen sizes.
6. **AI-Driven Exam Generation**: An innovative feature that uses OpenAI's capabilities to generate questions based on specified topics and difficulty levels. This tool aids instructors in creating diverse and challenging exam content.

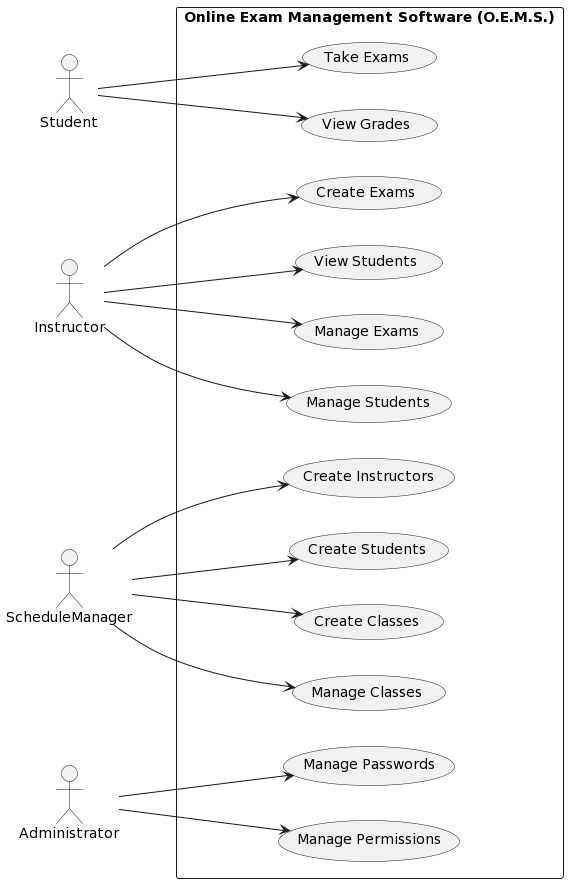
Class diagram overview for whole project





#### **Solution Strategy**

The development strategy focused on modular design. This approach facilitated iterative development and testing, enabling the team to incorporate feedback and make continuous improvements. Key strategies included:

* **User-Centric Design**: Emphasis was placed on creating intuitive interfaces for both instructors and students, reducing the learning curve and enhancing user experience.
* **Integration of AI Technology**: Leveraging AI for exam question generation, providing a novel approach to exam creation that augments the traditional question bank method.
* **Security Measures**: Implementing advanced security protocols, such as SSL encryption, secure login processes, and role-based access control to protect sensitive data and exam integrity.
  + The security measures therefore differentiated the respective user permissions for different user types. The use case diagram below portrays the different use case permissions for all different user types in the software.
  + 

### **5. UML Diagrams**

We have shown some UML diagrams in this document but the bulk of them reside in our technical manual. If you would like to view all UML that outline all the different methods and classes within the project, navigate to the project itself and find the Documents folder, this folder contains a PlantUML folder that holds the entirety of the UML files that were created based on this project.

### **6. System Requirements**

#### **Introduction**

Defining system requirements was a critical step in ensuring the successful development and deployment of the Exam Management Software. These requirements were categorized into broad and specific objectives, focusing on functionality, usability, and security.

#### **System Development tools**

The system was designed to be platform-independent and accessible via web browsers. Key requirements included:

* **Server**: Robust server capable of handling simultaneous connections and data processing.
* **Development Environment**: Eclipse IDE and Maven for software development and dependency management.
* **Technologies**: Java for backend development, HTML/CSS/JavaScript for frontend, and SQL for database management.
* **Third-Party Integrations**: OpenAI API for the AI-driven exam generation feature.

#### **Hardware and Software Requirements**

* Computer requirements
  + OS: Windows 10,11/ Mac Ventura
  + Browser: Google Chrome, Mozilla Firefox
  + Processor (CPU):
    - Intel Core i3 or equivalent AMD processor (or higher)
  + Memory (RAM):
    - Minimum 6 GB RAM

### 

### **7. Technical Specifications and Implementation**

#### **Technical Framework**

The technical framework of the Exam Management Software was designed to be robust and scalable. It included:

1. **Front-end Development**:
   * Utilized HTML5, CSS3, and JavaScript to create a responsive and accessible user interface.
   * Employed frameworks like Thymeleaf for frontend/backend interactions.
2. **Back-end Development**:
   * Built with Java, utilizing Spring Boot for a streamlined, convention-over-configuration approach to application setup.
   * Integrated with Maven for dependency management, ensuring that all libraries and frameworks were up to date and compatible.
3. **Database Management**:
   * Configured a MySQL database to store user data, exam questions, results, and other relevant information.
   * Implemented CRUD repositories and set up JPA repositories within the system to streamline data collection and storage processes.
4. **AI Integration**:
   * Incorporated OpenAI's GPT model to generate exam questions, providing an innovative alternative to traditional question banks.
   * Developed a custom interface for instructors to interact with the AI features, such as setting the topic, difficulty level, and number of questions.
5. **Security Measures**:
   * Using a CustomAuthenticationSuccess Handler to successfully redirect the proper user logging in to their desired homepage. (Instructor, Student, Schedule Manager, Administrator)
     1. This is implemented in SecurityConfig.java where the login page is prompted along with the logout feature. All of this is a part of the HttpSecurity Springboot Package.

#### **Implementation Process**

The implementation followed an agile process with the following phases:

1. **Requirement Analysis**:
   * Conducted interviews with potential users to gather detailed requirements.
   * Compiled a requirements document that served as a blueprint throughout the development process.
2. **System Design**:
   * Designed the system architecture with a focus on modularity and scalability.
3. **Development and Coding**:
   * Developed the application in iterative cycles, with each cycle introducing new features or improvements.
   * Conducted code reviews to maintain coding standards and ensure code quality.
4. **Integration and Testing**:
   * Integrated various system modules and tested their interactions.
   * Testing was largely done through use of the program itself.
5. **Feedback and Iteration**:
   * Implemented changes and improvements based on feedback in subsequent iterations.

### **8. Testing and Validation**

The testing and validation process for the Online Exam Management Software was comprehensive and multi-faceted, ensuring not only the functional correctness of the system but also its performance, user satisfaction, and compliance with relevant standards.

1. **Results Analysis:**
   1. *Functional Correctness*: Each test, such as those ensuring the correct functionality of instructor and student information management (e.g., setting and retrieving IDs, names, emails, passwords, and roles) was carefully analyzed against expected results. This included verifying the accuracy of data handling in the InstructorTest and StudentTest units, ensuring the integrity and security of personal and academic information within the system.
   2. *Performance Metrics:* The system was evaluated against performance goals, including its responsiveness and efficiency in handling operations like updating exam details, evaluating answers, and managing user roles. The performance of these critical functionalities, as tested in the J-Unit tests, was crucial for maintaining a smooth and effective user experience.
2. **User Feedback:**
   1. *User Experience Insights:* Feedback from both instructors and students who interacted with the system was gathered. This helped in assessing the user interface's intuitiveness and the overall usability of features like exam management, course enrollment, and account management.
   2. *Qualitative Feedback Methods:* Surveys and interviews were conducted to collect qualitative feedback. These methods provided deeper insights into user satisfaction and areas needing improvement, especially regarding the ease of navigating the system and the clarity of information presentation.
3. **Compliance Checks:**
   1. *Data Privacy and Security:* The system was rigorously checked for adherence to data privacy and security standards, especially in handling sensitive user information as seen in the Instructor and Student domain models.
   2. *Standard Compliance:* The software was also reviewed for compliance with educational and software development standards, ensuring it met the necessary criteria for a secure, reliable, and effective educational tool.

**8.1 Overview**

In the development of the Online Exam Management Software, rigorous testing and validation were crucial to ensure the functionality, reliability, and security of the system. This section outlines the testing and validation strategies employed, focusing on J-Unit Tests for the critical components of the system.

**8.2 Testing Strategy**

The testing strategy employed in the development of the Online Exam Management Software mainly involved unit testing, and user acceptance testing. This section focuses on the unit testing aspect, particularly the J-Unit tests.

**8.2.1 J-Unit Testing**

J-Unit tests were written to test individual units of code in isolation from the rest of the system. This helped in identifying and fixing issues early in the development process. The primary areas covered in J-Unit testing included:

*ExamServiceTest:* Testing the services related to exams.

*ExamTest:* Testing the functionalities of the Exam domain.

*InstructorTest:* Ensuring the correct functionality of the Instructor domain.

*StudentTest:* Validating the functionalities associated with the Student domain.

**8.3 Test Cases and Validation**

**8.3.1 ExamServiceTest**

The ExamServiceTest focuses on testing the ExamService class's methods, employing Mockito for mock objects and ByteBuddy for creating dynamic classes to simulate real-world scenarios. Each test method is designed to validate a specific aspect of the exam management process:

*testGetExamById():*

*Objective:* Validates if the correct exam is retrieved by its ID.

*Mockito Usage:* Uses Mockito to mock the ExamRepository and simulate the retrieval of an exam by ID.

*testUpdateExamName():*

*Objective:* Ensures the exam name is updated correctly.

*Mockito Usage:* Employs Mockito to mock the ExamRepository, simulating the process of finding and saving an updated exam.

*testUpdateDuration():*

*Objective:* Checks if the exam duration updates properly.

*Mockito Usage:* Utilizes Mockito for mocking the ExamRepository to test the update of an exam's duration.

*testUpdateStartTime():*

*Objective:* Verifies the functionality to update the exam's start time.

*Mockito Usage:* Mockito is used to mock the ExamRepository for testing the update of the start time of an exam.

*testUpdateEndTime():*

*Objective:* Tests if the end time of an exam is updated accurately.

*Mockito Usage:* Applies Mockito to mock the ExamRepository for validating the end time update functionality.

*testAnswers():*

*Objective:* Validates the exam answer evaluation logic.

*testUpdateExamDetails():*

*Objective:* Ensures that the exam details update functionality works correctly.

*Mockito Usage:* Uses Mockito to mock the ExamRepository, facilitating the testing of updating exam details for both existing and non-existing exams.

*testEvaluateAnswers():*

*Objective:* Tests the evaluation of user answers and scoring.

**8.3.2 ExamTest**

*testSetAndGetId():* Tests setting and getting exam ID.

*testSetAndGetExamName():* Validates exam name set and get operations.

*testSetAndGetDuration():* Checks the duration set and get functionalities.

*testSetAndGetStartTime():* Ensures start time is set and retrieved correctly.

*testSetAndGetEndTime():* Tests the end time setting and getting functionalities.

*testSetAndGetCourse():* Validates course association with the exam.

*testSetAndGetQuestions():* Checks the functionality to set and get exam questions.

*testSetAndGetSubmissionCount():* Validates the count of exam submissions.

*testFormatStartTime():* Tests the formatting of the start time.

**8.3.3 InstructorTest**

InstructorTest focuses on validating the functionality of the Instructor domain model in the system. Each test ensures that various aspects of an instructor's data can be correctly set and retrieved, reflecting accurate handling and storage of instructor-related information in the system.

*testInstructorId():* This test ensures that the instructor ID can be set and retrieved correctly. It is crucial for uniquely identifying each instructor in the system.

*testInstructorRoles():* Tests the ability to assign and retrieve roles to an instructor. This is important for defining access and permissions within the system.

*testInstructorFirstName():* Validates whether the instructor's first name is correctly set and fetched. This is a basic test to ensure the integrity of personal information storage.

*testInstructorLastName():* Checks if the instructor's last name is accurately set and retrieved, which is essential for maintaining accurate instructor records.

*testInstructorEmail():* Ensures that the instructor's email address can be set and retrieved correctly, which is vital for communications and identification in the system.

*testInstructorPassword():* This test verifies the setting and retrieving functionality of the instructor's password, a critical aspect for maintaining security and privacy.

*testInstructorUsername():* Tests the functionality to set and retrieve an instructor's username. This is essential for account management and authentication processes.

*testCreditsTaught():* Validates the functionality to set and retrieve the total credits taught by an instructor, which is an important metric for academic workload management.

*testCourses():* Ensures that the courses taught by an instructor can be correctly associated with and retrieved from the instructor record, important for course management and scheduling.

*testSetUser():* This test verifies the integration of the instructor's user account, ensuring that personal and login information is correctly linked and managed in the system.

**8.3.4 StudentTest**

StudentTest is designed to ensure the correct functionality and data integrity of the Student domain within the system. Each test case verifies a specific aspect of student data handling.

*testStudentId()*: Validates the ability to set and retrieve a student's ID, essential for uniquely identifying students within the system.

*testStudentFirstName():* Ensures that the student's first name can be correctly set and fetched, verifying the integrity of personal information storage.

*testStudentLastName()*: Tests the accuracy of setting and retrieving the student's last name, a crucial aspect of maintaining correct student records.

*testStudentEmail()*: Checks the functionality to set and retrieve a student's email address, critical for communication and identification purposes.

*testStudentPassword()*: Verifies the secure handling of student passwords, including setting and retrieving functions, essential for maintaining system security.

*testStudentUsername():* Ensures that the student's username can be set and retrieved accurately, a key component for account management and access control.

*testCreditsTaken():* Tests the functionality to set and retrieve the number of credits taken by a student, important for academic tracking and record-keeping.

*testCourses():* Verifies that courses enrolled in by a student are correctly associated with and retrievable from the student's record, a vital aspect of course management.

*testRoles():* This test ensures that roles can be assigned and retrieved for a student, which is important for defining their access and permissions within the system.

*testEnabled():* Checks the functionality to enable or disable a student's account, an important feature for account management and security.

*testUser():* Verifies the integration of the student's user account with their personal and academic information, ensuring a seamless and accurate record-keeping system.

*testToggleEnabled():* Tests the toggle function for enabling or disabling a student's account, a critical feature for account status management.

**8.4 Testing Conclusion**

The comprehensive testing of the Exam, Instructor and Student domain models through these J-Unit tests ensures a high level of data integrity, security, and functionality within the Online Exam Management Software. By rigorously validating each aspect of these models, the software maintains accuracy in handling instructor and student information, which is fundamental to the system's overall effectiveness and reliability.

### 

### **9. Caveats/Minefields**

**Introduction**

This section goes into all the problems faced while making the system. It explains how some problems got fixed with short explanations and gives more details about the problems that are still not solved.

#### **User Deletion and Editing**

* + - 1. **Problem**

At the beginning of the development of this system, we created a method to edit a user's information that was stored in the database. While this method would work, the way that we updated the users was to essentially overwrite the user with the same information or the information that was changed in the editing process. We used a similar method for the editing process for each of the other user types. While this didn’t matter for schedule manager and instructor user types, it would make the student user types drop all associations throughout the system. So course associations would be lost while editing the student.

* + - 1. **Solution**

We then had to develop a new method to update the student account user types. This method would use the same information that the edited model that was passed to the method. Then using that model it would find the existing user entity and simply update that existing account, keeping all known associations.

#### **Two Factor Authentication**

* + - 1. **Problem**

Two factor authentication is when trying to verify who you are to the system using a personalized code to an email, or phone number to verify your identity. When attempting to implement a form of two factor authentication, we struggled to find a method that enabled this functionality within our system.

* + - 1. **Solution**

Unfortunately we were unable to implement this functionality into our system. Finding a solution for this was something that we were tasked with rather early on and did not have time to loop back to incorporating this feature.

#### **Login Page**

* + - 1. **Problem**

Getting the login page to a working state took approximately around 4 weeks, if it took

* + - 1. **Solution**

Unfortunately we were unable to implement this functionality into our system. Finding a solution for this was something that we were tasked with rather early on and did not have time to loop back to incorporating this feature.

### **10. Required Classes**

**1. Domain Entities:**

**Administrator.java**: Likely needed to represent administrators in the system.

**Course.java**: To represent course information.

**Exam.java**: To hold data about individual exams.

**ExamDetails.java, ExamQuestion.java, ExamQuestionDisplay.java, ExamResult.java, ExamSubmission.java, ExamSubmissionEntity.java:** These are needed for creating, displaying, submitting, and storing exam questions and results.

**Instructor.java**: To represent instructors who can create and manage courses and exams.

**ScheduleManager.java**: To manage exam schedules.

**Student.java**: To represent students who take exams.

**2. Repositories:**

**AdministratorRepository.java, CourseRepository.java, ExamRepository.java, ExamSubmissionRepository.java, InstructorRepository.java, RoleRepository.java, ScheduleManagerRepository.java, StudentRepository.java, UserRepository.java:** These interface files are required for data access operations (CRUD) for their respective entities in the database.

**3. Services:**

**CourseService.java, EmailService.java, ExamQuestionService.java, ExamService.java, ExcelExportService.java, ExcelGeneratorService.java, ExcelParserService.java, InstructorService.java, ScheduleManagerService.java, StudentService.java, UserService.java:** Service files encapsulate business logic and are essential for processing data, sending emails, managing courses and users, generating and parsing Excel files, etc.

**4. Controller:**

Various controllers (e.g., **ExamController.java**, **InstructorController.java**, **StudentController.java**, **ScheduleManagerController.java**, etc.): These are crucial for handling HTTP requests, directing traffic, and providing endpoints for the application's functionalities.

**5. Configuration:**

**SecurityConfig.java**: For configuring security settings, including authentication and authorization.

**7. DTO (Data Transfer Objects):**

**CourseGradeDTO.java, ExamGradeDTO.java:** To transfer complex data structures between processes, used for reporting grades.

**8. User Registration and Management:**

**User.java, Roles.java:** Essential for user management and role-based access control.

**UserService.java, UserServiceImpl.java:** To handle business logic related to users.

**UserRegistrationController.java:** To manage user registration processes.

**9. Main Application: OnlineExamManagementSystemApplication.java:** The main application file that bootstraps and runs the Spring Boot application.

**11. Glossary of Terms for Online Exam Management System**

**OnlineExamManagementSystemApplication.java**

**CommandLineRunner beans:** Special beans in Spring Boot used for running logic after the application starts.

**Spring Boot's inversion of control**: A design principle in Spring Framework used for dependency injection and bean lifecycle management.

**Class Methods (OnlineExamManagementSystemApplication.java)**

**main(String[] args):** Launches the Spring application.

**setupRoles():** Initializes necessary roles in the database.

**setupDefaultUser():** Creates default user accounts for each role.

**createRoleIfNotFound(String roleName):** Ensures a role exists in the database.

**createUserIfNotFound(...):** Ensures a user exists in the database with specified details.

**Autowired Components (OnlineExamManagementSystemApplication.java)**

**PasswordEncoder**: Encodes user passwords.

**RoleRepository**: Handles CRUD operations for roles.

**UserRepository**: Manages user data persistence.

**AdministratorRepository**: Deals with administrator data.

**StudentRepository**: Handles student data persistence.

**InstructorRepository**: Manages instructor data.

**ScheduleManagerRepository**: Deals with schedule manager data.

**CustomAuthenticationSuccessHandler.java**

**onAuthenticationSuccess(...):** Customizes behavior post-authentication.

**OpenAIConfig.java**

**RestTemplate template()**: Configures a RestTemplate for OpenAI API interactions.

**SecurityConfig.java**

**passwordEncoder()**: Declares a bean for password encryption.

**configure(HttpSecurity http):** Configures web security for HTTP requests.

**configureGlobal(AuthenticationManagerBuilder auth)**: Sets up global security configurations.

**AdministratorController.java**

Methods for creating administrators, viewing admin homepages, account management, and CRUD operations for users and roles.

**CustomBotController.java**

Methods for interacting with a chatbot, including chatting, selecting topics, and quiz generation.

**ExamController.java**

Methods for managing the lifecycle of exams, including creation, editing, deletion, and review of submissions.

**ExcelController.java**

Methods for importing data from Excel files into the system.

**ExcelExportController.java**

Methods for exporting data, such as student information, to Excel files.

**InstructorController.java**

Methods for instructor-related functionalities, such as managing exams and editing information.

**ScheduleManagerController.java**

Methods for schedule management, including course and instructor management, and account details.

**StudentController.java**

Methods for student account management, course enrollment, and grade viewing.

**UserRegistrationController.java**

Placeholder structure for user registration management.

**Entity Classes**

**Administrator.java:** Entity representing an administrator.

**Course.java:** Entity representing a course.

**Exam.java:** Entity representing an exam.

**ExamDetails.java:** Class holding information for an exam.

**ExamQuestion.java:** Entity representing an exam question.

**ExamQuestionDisplay.java:** Class for displaying exam question details.

**ExamResults.java:** Class for storing exam results.

**ExamSubmissionEntity.java:** Entity representing an exam submission.

**Instructor.java:** Entity representing an instructor.

**Question.java:** Class representing a generic question.

**Roles.java:** Entity representing a user role.

**ScheduleManager.java:** Entity representing a schedule manager.

**Student.java:** Entity representing a student.

**User.java:** Entity representing a system user.

**Repository Interfaces**

Interfaces providing JPA repository methods for entities like **AdministratorRepository**, **CourseRepository**, **ExamRepository**, **InstructorRepository**, **RoleRepository**, **ScheduleManagerRepository**, **StudentRepository**, and **UserRepository**.

**Service Classes**

Classes providing business logic for entities, such as **CourseService**, **EmailService**, **ExamService**, **ExcelExportService**, **ExcelGeneratorService**, **InstructorService**, **ScheduleManagerService**, and **StudentService**.

**DTO Classes**

**ChatGPTRequest.java:** DTO for ChatGPT request data.

**ChatGPTResponse.java:** DTO for ChatGPT response data.

**CourseGradeDTO.java:** DTO for course grades data.

**ExamGradeDTO.java:** DTO for exam grades data.

**Message.java:** DTO for message information.

**UserRegistrationDto.java:** DTO for user registration data.

This glossary provides a detailed roadmap of the system's classes and components, indicating their roles and locations within the Java package structure. Each term is followed by a brief description of its responsibility or usage within the application, for more information please look in the technical manual.

### **11. Code Reusability**

**Introduction**

In building our system, we put a lot of thought into making it efficient and easy to expand. One key element we focused on was making sure our code could be reused in different parts of the program. We designed our classes and methods to work together smoothly, allowing us to use them again and again throughout the system. This approach helped us create a more organized and flexible system that works well across different modules and for different types of users.

**Class Reusability:**

Our program is structured for code reusability, primarily in the design and implementation of our various classes. Classes such as *Student.java, ScheduleManager.java, Question.java, Instructor.java, ExamSubmission.java, ExamResult.java, ExamQuestionDisplay.java, ExamQuestion.java, ExamDetails.java, Exam.java, Course.java, Chapter.java, and Administrator.java* are architectured to serve as reusable entities across different functionalities within the system.

Each of these classes represents a distinct entity within the program, exhibiting clear responsibilities and functionalities. They serve as foundational building blocks, allowing the instantiation and manipulation of objects across diverse modules within the system.

**Method Reusability for User Actions:**

Moreover, our program employs a cohesive set of methods that promote reuse, especially concerning user actions across different user types. These methods, strategically embedded within controllers, facilitate actions applicable to various user roles while maintaining a consistent structure.

For instance, in our AdminController.java, ScheduleManagerController.java, InstructorController.java, and StudentController.java, we harness the same set of methods such as *accountManager(), editingCurrentUser(), and saveCurrentUserEdits().* These methods empower users to manage their accounts efficiently, with modifications and adaptations made based on the specific privileges and functionalities tied to each user type.

Reusing these methods helps cut down on repeating the same code. It keeps our system strong and makes it easy to handle different things that different users need to do.

By embracing these design principles, our program maximizes code reusability, fostering maintainability, consistency, and scalability across various modules and user interactions.

### **12. Deployment/Maintenance**

**Introduction**

The act of Deploying a software system can be a very complicated and difficult task. There are many different ways to do so and each decision can lead to many complications or success. It's about more than just installing; it's about making sure everything runs seamlessly.

**Deployment of our system:**

Unfortunately, due to the time constraints during this semester, we encountered challenges in getting our system to the point of deployment to a server and conducting comprehensive testing.

We couldn't quite get our system onto a live server for testing, which put a damper on really putting it through its paces in the real world. We're totally aware that deploying it and giving it a thorough run-through is crucial for the full picture.

**Maintenance of our system:**

While we do not have our system deployed here are some maintenance tips to help if it is ever deployed.

**Backups and Recovery:** Regularly backing up data and creating recovery plans to help prevent any data loss or system failures.

**Code Refactoring:** Periodically refactor code to enhance readability, maintainability, and scalability.

**Documentation Updates:** Keep user guides, technical documentation, and code comments up-to-date to aid understanding and maintenance.

### 

### **13. Post-Mortem Analysis**

**Introduction**

This section will be an overall reflection of the project that will list some of the issues that arose later on that we did not have time to circle back around to and fix.

**Adoption of Design Patterns:** Implementing design patterns from the start would have increased the amount of code reusability, making the program more time and space complex, overall ending up in a more efficient program.

**Authentication and Authorization Improvements:** Granted authority is currently used in the project, switching to JWT would help tighten the endpoints to each respective user.

**Performance Optimization:** Implementing better algorithms in searching for users or generating tests would have been ideal and not take up much of the developers time..

**Frontend Frameworks:** Upgrading the interface would have been another thing that we could have done more, we provided a background for users but the layout to the original project is structured similarly at times in the navbar for each user.

**Testing and Continuous Integration:** Our approach primarily focused on unit and system testing. The J-Unit testing framework was extensively utilized to validate individual units of code, ensuring that each function performed as expected in isolation. This method was instrumental in identifying and rectifying bugs at the earliest stages of the development cycle.

However, one significant limitation in our testing strategy was the lack of integration testing. While unit tests are crucial for verifying the correctness of individual components, integration tests play a vital role in ensuring that these components work together seamlessly. The absence of integration testing in our workflow meant that we could not fully ascertain the interaction and data flow between different modules of the application. This oversight may have led to potential integration issues going undetected during the development phase.

While our approach to unit and system testing provided a solid foundation for software quality assurance, the absence of integration testing and a fully-realized continuous integration strategy were notable limitations. These areas, if addressed, could have significantly improved the overall robustness and reliability of the final product. Moving forward, incorporating comprehensive integration testing and robust CI practices will be crucial in enhancing the quality and efficiency of the software development lifecycle.

**Database Choices:** Using SQL seems to be very optimal, but perhaps using a different database such as MySQL Enterprise Edition when pushed to development.

**Security Enhancements:** This was certainly an aspect of the project that we fell behind. While that passwords are all encrypted and all the endpoints have been developed to the best of our ability we were unable to perform extensive security testing or improvements.

**User Experience and Accessibility:** We did a decent job of this while still falling behind in some aspects with it. Some of the UI does not have the best continuity while still being operational; it can sometimes be rather difficult to follow because some of the pages that do similar tasks do not follow the same UI conventions.

**Error Handling and Logging:** There are very little to no logging features for the next people to adopt this project and that will certainly hinder that group with understanding how our program is operating.

**Documentation:** We should have planned out how to structure our documentation more, and keep in mind the final documentation files that we needed. It came down to doing most of it last minute and rushing and it definitely hurt us at the end here not having a lot more of this done so we could have focused on fixing bugs in the program instead of rushing on the documentation.