**University of Leicester**

**School of Computing and Mathematical Sciences**

Dissertation

**----------------------------------------------------------------------------**

**A Tool for Managing Student Software Projects**

**----------------------------------------------------------------------------**

By

**Tharun King Antony**

A dissertation submitted in partial fulfilment

of the requirements for the degree of Computer Science BSc

CO3201 Computer Science Project

Supervisor: Dr Wentao Li

May 19th, 2025.

### DECLARATION

All sentences or passages quoted in this report, or computer code of any form whatsoever used and/or submitted at any stages, which are taken from other people’s work have been specifically acknowledged by clear citation of the source, specifying author, work, date and page(s).

Any part of my own written work, or software coding, which is substantially based upon other people’s work, is duly accompanied by clear citation of the source, specifying author, work, date and page(s).

I understand that failure to do this amounts to plagiarism and will be considered grounds for failure in this module and the degree examination as a whole.

Name: Tharun Antony

Signed:



# *Abstract*

Group-based student software project management raises a series of logistical problems for university staff, primarily concerning fair group allocation and assessment. This project developed a comprehensive web-based system to support convenors in streamlining student allocation, submission, peer feedback, and mark distribution in a structured and safe setting.

The application was developed in Java using Spring Boot, Thymeleaf, and MySQL as the framework, following the MVC architectural pattern. The primary features are a group allocation algorithm that considers student preferences to construct fair and compatible groups, a peer review system for obtaining feedback among the group members, and a mark distribution system that redistributes marks based on review details using a normalised multiplier. Access control was handled through Spring Security with role-based access and BCrypt hashing of passwords.

The group allocation algorithm was experimented with over multiple runs, confirming the robustness and consistency of the allocation algorithm. Manual and unit testing confirmed that the main functionality of the system satisfied the given requirements. All of the project objectives and the majority of the most wanted features were implemented successfully within the development timeframe.

Though the system is complete and covers the basic needs, a few future development plans include testing to find more intelligent group allocation using clustering techniques, file upload and cloud storage, better UI, and perhaps deployment to hosting services like Railway. Overall, the project offers a feasible and scalable solution to the issue of group-based student projects.

Contents

[**Declaration i**](#_Toc198483352)

[*Abstract* ii](#_Toc198483353)

[1 Introduction 1](#_Toc198483354)

[1.1 Group Allocation Problem 1](#_Toc198483356)

[1.2 Aims & Objectives 1](#_Toc198483357)

[2 Background and Related Work 3](#_Toc198483358)

[2.1 Classical allocation algorithms 3](#_Toc198483360)

[2.2 Clustering & Group Similarity Approaches 4](#_Toc198483361)

[2.3 Heuristic Algorithms 4](#_Toc198483362)

[2.4 Peer Review & Accountability Mechanisms 5](#_Toc198483363)

[3 Requirements Analysis 6](#_Toc198483365)

[3.1 Functional Requirements 6](#_Toc198483366)

[3.2 Non-Functional Requirements 8](#_Toc198483367)

[4 Design and Implementation 9](#_Toc198483369)

[4.1 System Architecture Overview 9](#_Toc198483370)

[4.2 Technology Stack 11](#_Toc198483371)

[4.3 Key Design Choices 11](#_Toc198483372)

[4.3 Database Implementation 13](#_Toc198483373)

[4.5 Implementation Overview 15](#_Toc198483374)

[4.5.1 Presentation Layer 15](#_Toc198483375)

[4.5.2 Service Layer 17](#_Toc198483376)

[4.5.3 Model Layer 19](#_Toc198483377)

[4.6 Group Allocation Algorithm 21](#_Toc198483378)

[4.6.1 Student preferences & Compatibility scoring method 21](#_Toc198483379)

[4.6.2 Heuristic Allocation Algorithm 23](#_Toc198483380)

[4.7 Mark Distribution System 27](#_Toc198483381)

[4.7.1 Submissions & Peer Review 27](#_Toc198483382)

[4.7.2 Adjusted Mark Calculation 27](#_Toc198483383)

[4.8 User Interface and Frontend 29](#_Toc198483384)

[5 Testing and Evaluation 39](#_Toc198483386)

[5.1 Overview of Testing Approach 39](#_Toc198483387)

[5.2 Manual Testing 39](#_Toc198483388)

[5.3 Unit Testing 41](#_Toc198483389)

[5.4 Testing Group Allocation Algorithm 42](#_Toc198483390)

[5.4.1 Methodology 42](#_Toc198483391)

[5.4.2 Results 42](#_Toc198483392)

[5.4.2 Comparing results to random allocations 44](#_Toc198483393)

[6 Critical Appraisal 45](#_Toc198483395)

[6.1 Critical Summary 45](#_Toc198483396)

[6.2 Discussion of Context 46](#_Toc198483397)

[6.3 Personal Development 46](#_Toc198483398)

[7 Conclusion 48](#_Toc198483400)

[References 49](#_Toc198483401)

[Appendices 51](#_Toc198483402)

[Appendix A – SecurityConfig.java 51](#_Toc198483403)

[Appendix B – Unit testing results 52](#_Toc198483404)

[Appendix B.1 – Password testing results 52](#_Toc198483405)

[Appendix B.2 – Student compatibility testing results 52](#_Toc198483406)

[Appendix B.3 – Group compatibility testing results 52](#_Toc198483407)

[Appendix C – Group Allocation Algorithm tests 52](#_Toc198483408)

[Appendix C.1 – Allocation algorithm testing data 53](#_Toc198483409)

[Appendix C.2 – Average group compatibility score of random allocation 58](#_Toc198483410)

**List of Figures**

Figure 1 - System Architecture Diagram showing the layered design of the application and the interaction between components. ............................................................................................. 9

Figure 2 - Entity Relationship Diagram (ERD) showing structure and relationship of the system's database ................................................................................................................................. 14

Figure 3 - Class diagram showing relationship between the controller and service layer ............... 16

Figure 4 - Class diagram showing the relationship between the service and repository layer ........ 18

Figure 5 - Class diagram showing the relationship between model classes .................................. 20

Figure 6.1 - Flowchart of group allocation algorithm (stage 1) ..................................................... 26

Figure 6.2 - Flowchart of group allocation algorithm (stage 2) ..................................................... 26

Figure 7 - Flowchart showing logic behind mark distribution system ........................................... 28

Figure 8 - Login page ................................................................................................................. 30

Figure 9 - Student home page ................................................................................................... 30

Figure 10 - View project details page (student) ........................................................................... 31

Figure 11.1 - View preferences page .......................................................................................... 31

Figure 11.2 - Set or edit preferences page .................................................................................. 31

Figure 12 - Give peer review page .............................................................................................. 32

Figure 13 - Convenor home page ............................................................................................... 32

Figure 14 - Convenor create project page ................................................................................... 32

Figure 15 - Assign students to project page ................................................................................ 33

Figure 16 - Group allocation page ............................................................................................. 33

Figure 17 - Convenor list of projects page ................................................................................. 34

Figure 18 - Convenor view project details page .......................................................................... 34

Figure 19 - Convenor edit project page ...................................................................................... 35

Figure 20 - View submissions and return mark page ................................................................... 35

Figure 21 - Convenor view peer review and adjusted marks ........................................................ 36

Figure 22 - Simplified Security Configuration Class..................................................................... 38

Figure 23 - A line graph showing the average group compatibility score across 10 allocations ....... 43

Figure 24 - Line graph showing average algorithm vs random allocation group compatibility score ....................................................................................................................................... 44

**List of Tables**

[Table 1 - Table of Student Preference Survey Questions 21](#_Toc198481675)

[Table 2 - Manual test and results 39](#_Toc198481676)

# Chapter 1

# Introduction

## 1.1 Group Allocation Problem

Group work is an essential part of a software project and education. It helps students develop key skills such as teamwork, communication and technical collaboration skills, which are essential when working in this industry. However, a challenge many staff face is organising students into effective groups to ensure maximum satisfaction and productivity. When left up to random or manual allocation, it can lead to a disparity between the groups. There may be groups with mismatches in skills, working styles or communication. This can lead to dissatisfaction, underperformance and overall negative impact on the project.

## 1.2 Aims & Objectives

I aim to create a secure website which allows convenors to allocate students into groups effectively using a list of preferences provided by the user. I want to provide the students with a survey to get their personal skills and preferences when collaborating with others on a group project.

Another aim is to implement a peer review system which allows students to give reviews to one another after a group project. This can help the convenor to see who contributed the most when it comes to marking their project. In conjunction with this, I want to develop a mark distribution system that will take a student’s peer review score and automatically calculate a mark based on contribution

1. To polish up on and learn the programming skills I will be using to implement features in my project
2. Familiarise myself again with Java, Spring Boot, MySQL and other general coding practices.
3. Learn how to use Thymeleaf and CSS to create web applications
4. Research algorithms that can help me develop an allocation system for students
5. To set up a database to store all the relevant data for users, projects, groups, etc and have an entity relationship between the tables.
6. To create a web application using the MVC framework to provide an interface for all the operations
7. To develop a secure, role-based login and registration system for students and convenors
8. To develop an algorithm that will automatically allocate students into groups based on preferences
9. To provide convenors with a way to create view projects, assign students and allocate groups
10. To implement a peer review system to allow students to leave reviews on other students based on how much they think they contributed to the project.
11. Develop a mark distribution system which gives students a fair adjusted mark based on peer review

# Chapter 2

# Background and Related Work

In this section, I will be exploring different algorithms which may be helpful in developing an algorithm to assign students to groups based on preferences and group constraints, such as group capacity. I will be reviewing well-established methods and considering their relevance to my project to see if I can implement them into the group allocation in an academic setting.

## 2.1 Classical allocation algorithms

One of the most famous foundational problems in allocation is the Stable Marriage Problem proposed by Gale and Shapley in 1962 [1]. The stable marriage problem is stated as if: given n men and n women, where each person has ranked all members of the opposite sex in order of preference, marry them so that there are no two people of opposite sex that would rather marry each other than their current partner. Although it was originally created to solve college admissions, it has given me insights into preference-based matchmaking. The Gale-Shapley algorithm, the algorithm which they came up with to tackle the college admissions problem, iterates repeatedly with one side proposing and the other accepting the most favourable offer, slowly getting rid of any unstable pairings. Each iteration, the accepting side will be given proposals from the other side and will pick the best option available.

The stable Marriage Problem assumes a one-to-one matching structure, where each person is matched up with one other person. When assigning groups in a project, the situation is more complex, as a group will usually have more than 2 students, and there will be other constraints such as group capacity. This led to researchers building on the Stable Marriage Problem, taking the principles of it and adapting it into a more flexible framework, such as the Student-Project-Allocation problem [2]. The SPA problem doesn’t just consider the preferences of students, but also other constraints such as the project capacity and even lecturer preferences. There have been extensions of the Gale-Shapley algorithm which were proposed to tackle this problem, balancing both sides' preferences while ensuring that no project is oversubscribed.

Another traditional method that can be used in allocation problems is the Hungarian Algorithm[3]. Originally developed to solve the assignment problem, the Hungarian algorithm finds an optimal one-to-one mapping between agents and tasks based on minimising a total cost or maximising a total score. In the case of group formation, I could possibly adapt it by creating cost matrices based on compatibility scores between students. But, like the Stable Marriage Problem, the Hungarian Algorithm is most suited to one-to-one mappings. Its application to group-based or many-to-many issues would require a lot of modifications, and even then would not necessarily consider dealing with more qualitative aspects like leadership receptiveness or teamworking style, which are also very important within student teams.

The classical algorithms can provide a strong theoretical guarantee, but they also don’t have the flexibility to form groups in real-world situations where there can be incomplete preferences or preferences which inconsistent or not quantifiable.

## 2.2 Clustering & Group Similarity Approaches

As opposed to assignment-based models, clustering aims to put individuals into groups based on overall similarity rather than satisfying exact preferences. K-means clustering is a technique that is widely used, which partitions individuals into a predetermined number of clusters in such a way that everyone within a cluster is as similar to each other as possible [4]. Applying this group allocation, students can be represented using a feature vector (skills, working hours, etc.) and then grouped based on the distance between each student, where they are grouped with students closest to them.

The advantage of clustering techniques is that they are able to handle multidimensional data and provide groups that are consistent in the chosen features. Cluster algorithms typically do not guarantee that the resulting groups will be well-balanced in leadership distribution, skill diversity, or work style. Clustering might produce overly similar groups that would make group dynamics weaker and limit the occurrence of complementary skills in a project team [5]. Furthermore, clustering algorithms are sensitive to initial conditions, particularly K-Means, such that random initialisation can provide different outcomes unless tightly controlled.

Clustering also assumes that all features are equally important as each other, which means that unless there is manual weighting and implementing it is a complex task in itself.

## 2.3 Heuristic Algorithms

Acknowledging the limits of both classical matching and clustering techniques, many recent studies have turned to heuristic-based methods for student group formation. By definition, heuristics look for satisfactory rather than ideal answers, so compromising mathematical guarantees for practical performance, interpretability, and flexibility.

Usually, heuristic group-building techniques define a scoring mechanism that aggregates several compatibility elements. For example, Points might be given for students with similar working hours, complementary skill sets, or leadership ability; groups are created to maximise these scores while fulfilling group size restrictions [9]. Furthermore, heuristic approaches are better suited to real-world learning environments where not all students submit complete preference forms, since they can tolerate incomplete or inconsistent preference data.

Studies show that a heuristic approach is really effective for balancing fairness and functionality. Systems based on heuristics can cluster students who prefer similar working hours together [10] and enforce policies such as ensuring each group consists of at least one student ready to lead. In addition to this, heuristics can be made to give different criteria top priority depending on the course of action: for instance, emphasising technical skill balance in software development projects or leadership diversity in leadership-heavy projects [11].

The adaptability of heuristics is one of their main benefits. Heuristic models, as opposed to strict algorithms, can be changed to fit different learning environments without requiring a total redesign. However, heuristic algorithms don’t ensure globally optimised outcomes, and their effectiveness is largely dependent on how the scoring function and allocation process are designed.

## 2.4 Peer Review & Accountability Mechanisms

Peer review is another important part of group management as it allows a convenor to monitor an individual's contributions to the group project and allows them to reward each student fairly. Peer Review has been widely used as a means to help and encourage student accountability within the group. It has been demonstrated that structured peer evaluation systems, in which students rate their teammates according to well-defined behavioural criteria, reduce problems like social loafing and free-riding [5], [6].

Despite this, it is very difficult to design an effective peer review system. Studies have shown that if not done properly, there may be discomfort around giving negative feedback, there may be evaluator bias, and personal conflicts among students[7]. They even recommend normalising scores in order to check for a harsh or generous review, and some even suggest using blind reviews to try and reduce bias. Although there was consideration of peer review in the early stages of this project and the possibility of extending peer assessment, a proper, working and fair peer assessment system is a considerable technical and social challenge, one that typically requires planning and several rounds of testing each step in order to develop.

# Chapter 3

# Requirements Analysis

This section goes over the key requirements that helped me develop my system. It talks about the essential features of my system needed to meet the objectives I set. Functional and non-functional requirements are separately presented.

## 3.1 Functional Requirements

Functional requirements are what define what the system is meant to do. It describes all the features and needs for its users. Since it is made for 2 types of users, students and convenors, it has been designed with 2 main user roles in mind.

**The Functional Requirements for Students:**

1. Preference Submission (essential): After logging in, the students should be able to take part in a survey which will ask about their preferences and skills that will later be used in the group allocation. These preferences include:

* Preferred working hours
* Willingness to lead the group
* Preferred teamwork style (whether they prefer working independently or collaboratively)
* A self-assessed technical skills rating and more.

1. View Assigned Project (essential): The student should be able to see all the projects which the convenor has assigned them to. They should also be able to see details about the project, such as deadlines.
2. View Assigned Group (essential): Once the groups have been allocated by the convenor, students should be able to log in and view which group they have been placed in and who their group members are.
3. Submit Projects (essential): The students should be able to let the convenor know that they have submitted a project so the convenor can return marks.
4. Peer Review (recommended): The student should be able to submit a peer review on their group members once the project is submitted. It should be quantitative with an optional written justification that the convenor may see.

The aim here was to make the student side of the system simple. Students should not need any training or instructions to use the system properly.

**The Functional Requirements for Convenors:**

1. Project Creation and Management (essential): Convenors should be able to create new projects within the system, specifying details such as project titles and maximum group size. They should be able to edit the project and change details after creation.
2. View Registered Students (essential): Convenors should have access to a list of all registered students and be able to assign them to projects and groups.
3. Auto Group Allocation (essential): Convenors must have a way to automatically assign students into groups using the preferences they set.
4. View Group Allocation (essential): After the students are assigned to groups, the convenor should be able to see the groups as well as the groups' compatibility score average between the students.
5. View Submissions (essential): Convenors must be able to see a list of submissions made by groups for a project and return a mark for each submission made.
6. Auto Mark Distribution (essential): Once the convenor has returned a group’s mark, the system should calculate an adjusted mark for each student in that group based on the peer reviews left for that group.

**System-Wide Functional Requirements:**

In addition to user-specific functionality, there were several system-wide features that needed to be in place to ensure the application worked reliably and securely.

1. Login and Registration (essential): There must be a login and registration system that recognises the role of the user when they log in and takes them to the corresponding home page. There must be an option when registering to choose the role the user is going to be, student or convenor.
2. Role-Based Access Control (essential)**:** Students should not be able to access any convenor-specific functionality, and vice versa. Pages should be hidden or blocked based on user roles after login.
3. Data Persistence (essential): All user details, students' preferences, project details and group assignments must be stored securely in a MySQL database.
4. Password Security (essential): Passwords must be encrypted using a secure method before being saved in the database to prevent the password from being exposed in the event the database is breached.
5. Web Interface (essential): The system should be accessible via a web browser through a localhost server, for now (could allow hosting using AWS in the future, not necessary)
6. Error Handling (essential): Basic error messages should be shown when there is an error.

These requirements were essential for making sure the system behaved properly, protected user data, and didn't allow accidental or malicious misuse.

## 3.2 Non-Functional Requirements

Non-functional requirements define how well the system should perform, rather than what it does. These requirements focus on usability, security, performance, portability, and maintainability.

1. Usability: The system should be simple and intuitive to use for both students and convenors. The layouts must be clear with easy-to-fill-in forms, and navigation should be obvious without the need for a user manual or training. Students and convenors are both developers, so they should be able to easily navigate around the system due to their experience in technology.
2. Security: A Strong focus should be put on security, especially around login, registration and role-based pages. Passwords should be encrypted using BCrypt, and pages protected by role-based authentication using Spring Security. Only users with correct roles can access protected pages.
3. Performance: The system should be able to handle a typical class of computer science students within a few seconds. It should scale properly without slowing down dramatically as the number of students starts to increase.
4. Portability**:** The system was developed and tested mainly on Windows 11, but it is designed to run on any machine with Java 17 and Gradle installed. No major platform-specific code was used.
5. Maintainability: The system should be well structured. Key parts, like the models and controllers, should be separated properly to allow for future changes without having to modify major parts of the code.

# Chapter 4

# Design and Implementation

The chapter will describe the technical details of how the system was designed and built. I will go over and explain the overall structure of the project, the technologies used, and the design choices made during the development and along with the reason why they were made. This chapter will also cover how the database was designed, the key components implemented, and how the group allocation algorithm works. Finally, it will look at the frontend interface, the security and the deployment of the application locally.

## 4.1 System Architecture Overview

The system was built using a layered architecture, following the Model-View-Controller principle, which is often seen in Spring Boot applications[14]. The structure separates the different responsibilities within the application, making it easier to manage, scale and maintain. There are four key layers which make up the system: these are the presentation layer, service (or business) layer, repository (or persistence) layer and database layer [12]. Each layer plays a specific role in how the data flows from the user to the database and back.

At the top is the presentation layer, containing the controllers responsible for handling HTTP requests from the user. For this project, the StudentController and ConvenorController handle routes and actions specific to the students and convenors, respectively. Controllers don’t deal directly with the database instead, they pass the request through the appropriate service class in the next layer.

The next layer down is the service layer, containing the main business logic of the application. This is where operations like saving preferences are done using StudentService. One of the main aims in my project is also handled here: the heuristic algorithm for grouping students. It is kept in the GroupService of this layer. Given more time, I would have liked to split the logic into smaller, more dedicated services, but the structure kept it understandable while still being modular.

Below this is where the repository layer is, which is responsible for interactions with the database. It uses JPA (Java Persistence API) to map entity classes like User, StudentDetails, and Project to database tables in MySQL. This layer also includes repositories like StudentRepository and ProjectRepository. These are interfaces which provide access to the entities without having to write raw SQL. This separation ensures that data access across the app is clean and consistent.

Finally, at the bottom is the database layer, where all the persistent data is stored in a MySQL database. This includes tables such as users, projects, student\_details and more. The structure of the database is shown more in the ERD later, but in short, it reflects the domain model built into the application.

All the layers communicate bidirectionally. For example, when a student submits preferences, the request will travel down through the controller to the service, then the repository and finally into the database. The response (can be a simple redirect or updated data) then travels back up through the same. This is shown in the system architecture diagram below, which outlines how the layers are connected and where each class sits in the application.

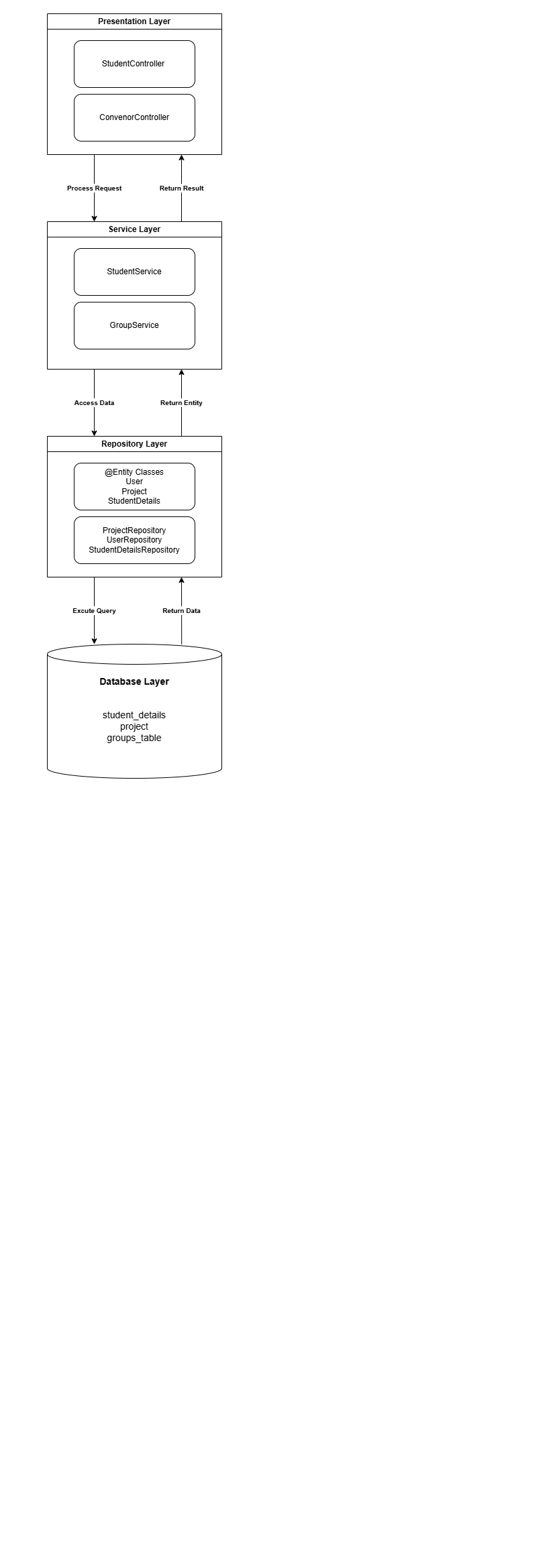


Figure 1 - System Architecture Diagram showing the layered design of the application and the interaction between components.

## 4.2 Technology Stack

The system was developed using a full stack of modern web development tools, with a heavy focus on Java and Spring Boot. I chose these technologies as I was already a little bit familiar with them, and they made sense for a system that needed to secure and test on localhost.

The backend of the system is built using Java 17 and the Spring Boot framework. Using the Model-View-Controller pattern, setting up a web application using Spring Boot was easy. Along with this, Spring Boot also supports JPA (Java Persistence API) and Spring Security, which I used for handling data persistence and user authentication. Annotations such as @Entity, @Service and @Controller helped maintain a readable code while keeping it modular.

The frontend of the system was built using Thymeleaf, which is a server-side HTML templating engine that integrates directly with Spring Boot [13]. This made it much easier to render dynamic content based on the user roles. An example of this is how the student and convenor get redirected to different home pages when logging on, and Thymeleaf makes managing this a seamless process.

For my database, I chose to use MySQL relational database to store all my persistent data, including user accounts, student and convenor details, student preferences, etc. I used MySQL Workbench to manage the database locally during development, allowing me to see the data being stored, taken, and the entity tables I created using JPA. I set a schema for the database manually in MySQL Workbench and then linked it to the entity classes in the application using JPA. Although Spring Boot supports auto-generation of schemas using JPA, I chose to define the structure myself for more control and consistency throughout testing.

Build and dependencies were handled with Gradle using the built-in wrapper gradlew so anyone can run the project without having to install Gradle separately.

For password encryption, I chose to use BCrypt hashing, which works well with Spring as it is built into Spring Security. This makes sure that the passwords aren’t stored as plain text and are secure in the database.

The entire project was developed using IntelliJ IDEA, with GitLab being used for version control and remote backups. Diagrams used in the project were created using online tools such as draw.io, and any screenshots provided were taken from running the application.

## 4.3 Key Design Choices

There were many important design decisions I had to make throughout this project, which impacted the way the project was built. Some decisions were based on technical reasons like framework compatibility, while some were driven by time constraints. In section I will explain why I took certain approaches and what this resulted in.

One of the most important decisions I had to make was to choose what algorithm I would use to make my allocation program. In the end, I chose to use a heuristic-based algorithm to assign students to groups. While the widely used algorithms, such as the Gale-Shapley and the Hungarian algorithm are strong, they are generally better suited for one-to-one matchmaking problems, like college admissions or pairing students with projects. In contrast to this, the system required group allocation, which is more complex as there are more constraints, and it needs to balance multiple student traits at once. While some parts of the stable matching were still useful, in particular when considering compatibility, a full implementation wouldn’t have been practical. The heuristic approach gave much more flexibility, as I could score students based on compatibility features such as working hours and work style.

For the backend of the application, I chose to use Spring Boot as it fit my needs, as well as the fact that I had some familiarity with it. As mentioned, it has high compatibility with JPA and Spring Security, which are both essential to my project. Since it supports dependency injection, it made defining service layers much easier and allowed for quick development using annotations. These advantages made Spring Boot a good fit for building a secure and scalable web app without needing to manually configure everything from scratch.

The frontend of the system uses Thymeleaf templates instead of JSP. I had originally started the project off with JSP pages, but JSP’s integration with Spring Security was limited and caused many issues during testing. On the other hand, Thymeleaf works seamlessly with Spring Boot and allows me to render dynamic pages based on the user's role. It also felt more modern, providing more powerful expressions such as *th:if* or *th:each,* leading to a much simpler set-up compared to JSP, which is old and only provides basic JSTL. Since students and convenors see different interfaces, Thymeleaf made it easier to load the correct content based on login credentials.

Due to time constraints, I chose to implement a simple peer review system where the students will give each other a score between 1 to 10, as well as leave an optional comment justifying their rating of a group member. For my mark distribution system, I normalised each student’s average peer review score relative to the group’s average, and used this to calculate a multiplier that is applied to the group’s overall mark and adjusted mark for each student.

I also had to resort to manual testing the system instead of writing full unit tests due to time constraints, I couldn’t go and test each small part of the code. While there are future plans to look into grouping using K-means clustering to try and achieve a better grouping system, I ensured that I prioritised the functionality of my system now to meet the main requirements.

## 4.3 Database Implementation

For my database, I used a relational schema implemented in MySQL. It contains all the core data needed to manage users, student preferences, projects, group allocations, peer reviews, and submissions. At the start of the project, I manually created the schema using MySQL Workbench. Later, when I implemented the entity classes in the application using JPA annotations like @Entity and @Id, it automatically mapped the model classes to the existing database and its structure.

The most important table is the user table, which stores the login credentials and personal information for both students and convenors. There is a role field which is used to identify the user as a student or convenor. The user table is a parent table and is connected to two child tables: student\_details and convenor\_details, which are used to store role-specific information.

Each student has a new student ID in the student\_details table, which is further linked to a student\_preference table containing information used during the group allocation, such as leadership preference, technical skills, communication skills, working hours, etc. These are the fields used to calculate the compatibility scores when forming groups. The student preferences can be accessed using the student ID from the student\_details table.

The project table stores all the information about the group projects created by the convenor. It includes a title, description, deadline, and group capacity. Each project is linked to one convenor, and each project can have multiple students assigned through the project\_students table.

The groups\_table stores all the groups formed during the allocation, and the membership of the students to the groups is tracked through the group\_member table. Each group is tied to a specific project. Group memberships are also linked to a submissions table, which stores a submission from the group. This table records whether the work was submitted, which group submitted, for which project, when it was submitted, and the final group mark.

The peer\_review table stores the peer review data submitted by students after the project deadline. It records the reviewer, the reviewee, the group and project they belong to, a rating out of 10, and an optional comment. These peer reviews are then used to calculate an adjusted mark stored in the mark table. The mark table stores the original and adjusted mark, and for which project and student it is for.

All relationships in the database are handled using foreign keys. For example, student preferences references the primary key of student\_details, project\_students forms a many to many between projects and students. This design aims to eliminate any data redundancy and keep the information connected in a normalised way. The diagram below shows the final database schema, with all primary keys, foreign keys, and table connections illustrated.

A computer screen shot of a computer

AI-generated content may be incorrect.

Figure 2 - Entity Relationship Diagram(ERD) showing structure and relationship of the system's database

## 4.5 Implementation Overview

This section explains how the system was built, going through key classes and components in each layer of the application. I will be exploring how the files are structured and how they follow the standard Spring Boot architecture, with the classes organised into controller, service and model layers. This design allows for future changes to be easier to manage as it separates concerns and keeps logic modular. The diagram below shows the major classes used in the application and how they interact with each other.

### 4.5.1 Presentation Layer

The presentation layer of my system is made up of several controllers that handle the HTTP requests sent by the client and the user interactions. Each controller is mapped to specific routes and functions depending on whether the user is a student or convener. Using the @controller annotations and Spring MVC to manage routing, the system can pass data from the frontend and backend through the models. As shown in Figure 3, each controller connects to the relevant services, which helps separate concerns and keep the code modular.

StudentController handles key student-related actions, such as loading the student homepage upon a successful login. On top of this, it manages the display of projects on the student's side. This includes showing a list of their projects, a page displaying project details, and handling the submission of a project. Additionally, it ensures that students can only access content relevant to their assigned groups and submissions, maintaining proper access control.

The PreferencesController is responsible for managing the student preference workflow. It renders the necessary web pages that allow students to create, edit, and view their project preferences, and handles the submission and storage of these preferences.

The ProjectController and ConvenorController together manage the convenor-facing functionality of the system. The ProjectController oversees the creation, deletion, and general management of projects, including displaying lists of students assigned or unassigned to specific projects and allocation group formation via the allocation algorithm implemented in the service layer. Meanwhile, the ConvenorController enables convenors to view project submissions from student groups and assign or return marks for those submissions.

User login and registration are handled by the LoginController and RegisterController, respectively. These provide the user with a login and register page and work with Spring Security to ensure authentication. The HomeController acts as the front page, which prompts any users who have yet to log in to head to the login page.

A screenshot of a computer screen

AI-generated content may be incorrect.

Figure 3 - Class diagram showing relationship between the controller and service layers

### 4.5.2 Service Layer

The service layer contains the logic which is used to connect the controllers to the repositories, acting as a middleman between the frontend and backend. It is responsible for processing data, applying any business rules and preparing responses for the presentation layer. By annotating each service with @Service it allows the system to keep logic separate and make it easy to maintain. You can see from the service layer diagram (Figure 4) how each service connects to the correct repository, showing that all business logic stays separate from data access.

The StudentService handles small tasks like retrieving information about the student. This includes getting any projects the student is doing, getting groups they are a part of, etc. StudentPreferencesService manages the conversion of preference form data into database entities and retrieves existing preferences for editing.

The ProjectService is used to manage everything to do with the projects in the system, like creating, updating or retrieving a project from the database. The most important part of the service layer is the GroupService, as it is responsible for containing the algorithm used to assign students to groups. It takes the student preferences from StudentPreferencesService and uses them to calculate a compatible group. It also contains other small functions.

The SubmissionService, PeerReviewService and MarkService all work together to handle the flow of returning marks to students. SubmissionService and PeerReviewService consist of functions used to save and retrieve submissions and peer reviews. MarkService holds the logic for calculating the adjusted mark based on peer reviews, and letting the convenor override the adjusted mark if needed

The RegistrationService handles the registration process, including role assignment, password encoding, and saving new users. Other supporting services include CustomUserDetailsService for Spring Security and DataInitializer, which was used during development to insert test data.

**A diagram of a computer

AI-generated content may be incorrect.**

Figure 4 - Class diagram showing the relationship between the service and repository layesr

### 4.5.3 Model Layer

The model layer contains all the entity classes that are mapped directly to the database. The most central class is the User. The contains all information regarding the client using the program. This includes the login details as well as the user's role, being a student or convenor. This is also linked to either StudentDetails or ConvenorDetails, depending on their role.

StudentDetails holds data specific to the user role, student, which can be a new student ID independent of the user ID, student preferences, etc. It is also linked to the StudentPreferences class, which holds all of the student's preferences, like preferred working hours or prior experience.

The ConvenorDetails stores data that is specific to convenors, such as projects they are managing and a Convenor ID.

Project class stores information about the details of a project, groups and students in a project. The students assigned to a project will be tracked using the ProjectStudents table, which isn’t a model but created using a @JoinTable annotation using JPA.

Groups class stores the groups created by the convenor. The membership of students is handled by the GroupMember class, which stores a list of students and what project that the group is in. After groups are allocated, the Submission class is able to track submissions, which have details about a submission made, such as submission times and marks for that submission.

A diagram of a computer program

AI-generated content may be incorrect.

Figure 5 - Class diagram showing the relationship between model classes

## 4.6 Group Allocation Algorithm

### 4.6.1 Student preferences & Compatibility scoring method

Before students can be allocated into the right group, the system first needs a way to determine which students will work well together. In order to find this, I made a custom compatibility scoring method. This method takes and compares two students' preferences and returns a score between the two students. This score is then used by the group allocation algorithm to create groups with students who have a high compatibility score.

To calculate this score, we take the 8 questions I asked the student and quantify them. The 8 questions asked of the user were :

Table 1 - Table of Student Preference Survey Questions

|  |  |  |  |
| --- | --- | --- | --- |
|  | Question | Answer Options | Score Type |
| 1 | Preferred working style? | 1: Independent, 2: Collaborative, 3: Flexible | Single choice (numerical) |
| 2 | Preferred working hours? | 1: Morning, 2: Afternoon, 3: Evening,  4: No preference | Single choice (numerical) |
| 3 | Confidence in technical skills? | 1-5 scale (1=Very low, 5=Very high) | Scale (1-5) |
| 4 | Confidence in communication skills? | 1-5 scale (1=Very low, 5=Very high) | Scale (1-5) |
| 5 | Leadership preference? | 1: Prefer to lead, 2: Prefer to support,  3: No preference | Single choice |
| 6 | Deadline approach? | 1: Finish early, 2: Finish close to deadline | Single choice |
| 7 | Teamwork experience level? | 1-5 scale | Scale (1-5) |
| 8 | Prior experience with this project type? | 1: Yes, 0: No | Binary |

After getting the student preferences, the method calculates a score from 0 to 1, where 1 means the pair of students has perfect compatibility and 0 means the students have no compatibility. The preferences have also been weighted slightly to give more importance to certain preferences. This approach aligns with methods that model preferences as weighted graphs to optimise group satisfaction [16].

To calculate the score, every pair of students starts at 0, and the system takes the 2 students' preferences and compares them.

Working style is weighted at 15%. The method compares to see if the students' preferred working style matches, and if they do, then it adds +0.15 to the compatibility score. If the student is flexible, then it automatically counts as a match to the other student.

Working hours is weighted at 10%. Works the same as working style, where if they match on working hours, it adds +0.10 with option 4: anytime being an automatic match.

Technical Skill is weighted at 15%. For technical skill, the closer a pair of students are in skill, the higher the score they receive. This is calculated by doing :

(1 - |Skill difference| / 4) \* 0.15

An example of two students with a skill ratings of 4 and 3 :

(1 - |4-3| / 4 ) \* 0.15 = (1- ¼ ) \* 0.15

= (0.75) \* 0.15

= 0.1125 (final score added to compatibility score)

Communication Skill is weighted at 10%. With communication skills, the method works to pair students who have low communication skills with students who have high communication skills to ensure that the group is communicating throughout the project. To achieve this, it combines the student's communication skills and compares them to an ideal score of six. The closer it is to that score, the higher they are rewarded.

Example of two students with skills of 4 and 1:

Combined skill = 4 + 1 = 5

Difference = |6 – 5| = 1

Score = 0.1 – (0.02 \* 1) = 0.08

Leadership Preference is weighted at 15%. This aims to pair up students who like to lead with students who prefer to support, with students who have no preference not being affected by this. If one is a leader, the other is a supporter, adds +0.1 to the score.

Deadline Approach is weighted at 10%. Students prefer to work with similar pacing (e.g. finish early or just before the deadline), they get an extra 0.10.

Teamwork Experience is weighted at 15%. The teamwork experience is calculated using the same method as communication skills to match up students with different levels of teamwork experience

Prior Experience is weighted at 10%. If one student has prior experience and the other doesn’t, it adds +0.15. If both students have prior experience, then it doesn’t add anything. This is done to group students who have experience doing this project with those less experienced, so they can help guide the group.

### 4.6.2 Heuristic Allocation Algorithm

The main feature of my program is the group allocation algorithm. After my background research and taking into account constraints, I decided the best way to implement a group allocation algorithm is to use a heuristic approach. I made a custom-built heuristic design to create fair and balanced groups based on student preference and skills. Compared to one-to-one matching-making algorithms like Gale-Shapley, a heuristic is favoured, as I am required to create small teams. To achieve this, I looked at the compatibility between multiple students and not just a pair.

The process starts once the convenor clicks a button to assign students to groups. The algorithm first checks if there are any students assigned to the project to put into groups. If no students are doing the project, then the system throws an error with an accompanying error message. Otherwise, it iterates through the list of students to check if they are already in a group for that project and creates a list of students who still need to be assigned to groups.

Then it uses the number of students who still need to be assigned to groups and the group capacity to calculate the number of groups needed. Students are then split into two groups: those who have submitted their preferences and those who haven’t.

Students with preferences are then shuffled to create randomness. This is done to ensure the students aren’t processed the same way each time. If they were to be organised into groups in order of ID or alphabetical order, it would create multiple problems. Firstly, it might create systematic bias that will favour students who are always first, as they will always have more group options available, while students who are later on will have fewer placement opportunities. Another reason to use randomness is to ensure that the algorithm isn’t stuck in sub-optimal allocation patterns. If it always processed students in the same order, then the same group formations may occur repeatedly. The shuffle allows there to a wide range of group allocation possibilities.

After they are shuffled, the system then takes the students and calculates a compatibility score for every student pair based on their preferred working hours, work style, teamwork, etc.

It then selects one student for each group to be the “seed”, which just means that it will place a student into each empty group to compare their compatibility to the unassigned students waiting to be assigned. The seed students are chosen using the leadership preference, where they choose if they like to lead, follow someone else's lead, or have no preference.

Naturally, the leaders and followers tend to have a higher compatibility score, so it assigns all the students who like to lead first before randomly assigning the rest of the seed students.

From there, the remaining students with preferences are added to the group one by one, where they score the highest in terms of compatibility. This will continue until all the students with preferences are exhausted, and after that, any students without preferences are simply assigned to available groups in a round-robin way.

Finally, the database is updated with the newly formed groups, and the convenor can view them on their view projects page. This algorithm employs heuristic properties as it makes the locally optimal choice at each step, but doesn’t backtrack and look at students who have already been assigned. It also uses the compatibility score as an approximation of what the preferred group for the student will be.

The algorithm also does basic group balancing to avoid a big difference in group sizes. It does this by first calculating the optimal number of groups for the project based on the number of students assigned. When placing students, it checks the size of the group compared to the max size and gives a slightly higher score to smaller groups. This makes it so that if multiple groups are suitable for the student, it will be more likely to place them in a group with fewer members. This ensures there is a fair balance of students between the groups while still maintaining high compatibility.

Additional logic was added to avoid creating new groups unnecessarily if existing groups still have space. This helps reduce wasted group slots and ensures that students without preferences are still distributed fairly, even if they’re added late.

Figure 6 below shows a flowchart of the logic:

A row of rectangular white rectangular objects

AI-generated content may be incorrect.

Figure 6.1 - Flowchart of group allocation algorithm (stage 1)

A diagram of a diagram

AI-generated content may be incorrect.

Figure 6.2 - Flowchart showing the steps of the group allocation algorithm from start to finish (stage 2)

## 4.7 Mark Distribution System

The system features a simple but effective mark distribution system that adjusts each group member's mark based on the peer reviews they receive. This part goes over the process of this and explains the intricacies behind it.

### 4.7.1 Submissions & Peer Review

The submission system lets student groups notify the convenor once their project is completed. Each group can submit their project using a form, which logs the submission time and marks it as submitted. Convenors can then view all submitted projects and return a mark for each one. At the moment, the system does not take in actual project files, but that could be explored in future improvements.

Once the students submit their project, they can leave peer reviews for the groups they worked with. Each student rates the others in the group on a scale of 1 to 10, as well as optionally leave a comment explaining their rating. These reviews are stored in the database and later used to calculate adjusted marks.

There are also deadlines set for both peer reviews and project submissions. The project can be submitted whenever, but will be marked as late if past the deadline, and the submission date/time will be shown to the convenor when they go to return marks. Students only have up to 3 days after the deadline of the project to submit or edit their peer review. This helps maintain a clear structure and keeps the marking process fair and consistent.

### 4.7.2 Adjusted Mark Calculation

After the 3-day peer review return period has the convenors can return a mark for each submission made for that project. Once the marks have been returned by the convenor, the system calculates adjusted marks for each student. This is done by comparing each student’s average peer review score to the average score across their group. A multiplier is calculated using the following formula:

To prevent any extreme adjustments in marks, the multiplier is clamped between 0.9 and 1.1, meaning no student can receive less than 90% or more than 110% of the original mark. The adjusted mark is then calculated as :

Figure 7 below shows the logic behind the method in the MarkService used to calculate the adjusted marks for the students.

**A diagram of a diagram

AI-generated content may be incorrect.**

Figure 7 - Flowchart showing logic behind mark distribution system.

The system was designed to be fair and simple, encouraging honest reviews without causing big swings in marks. It iterates through the list of groups for the project and gets all relevant data. Once it has got everything, it checks if there are enough peer reviews to run the algorithm, and only runs if the number of peer reviews given is more than 50% of the expected number of peer reviews. This is to stop biased marks and unfair adjustments being given if only one student returned peer reviews.

Example:  
If a group received a mark of 70, and a student scored 9.0 while the group average was 7.5:

Multiplier > 1.2 = 1.1 (clamped).

Once adjusted marks are calculated, the convenor can view them alongside the peer reviews from that group. If needed, they can override the final mark for any student. This gives convenors the final say in case of suspected bias or unusual scores, keeping the process both fair and flexible.

## 4.8 User Interface and Frontend

In this section, I will show some of the important web pages and explain their purpose. The user interface was built using Thymeleaf, which allowed me to create dynamic, role-based pages for students and convenors. It also integrates Spring Security, which allows me to restrict pages using roles, and if the user isn’t signed in, it redirects them to the login page.

The Login Page

This is the page the user gets redirected to if they try to access a restricted web page. This is also the login page, which all users must use to log in to use the system.

A computer screen shot of a building

AI-generated content may be incorrect.

Figure 8 - Login page

Student Homepage

This is the home page for students. It provides a list of the projects they are in and has a navigation bar at the top of the page to easily navigate around the website.

A screenshot of a computer

AI-generated content may be incorrect.

Figure 9 - Student home page

Student View Project Details

Shows the details of the chosen project, such as the project title, as well as whether they have submitted and been given a mark.

A screenshot of a computer

AI-generated content may be incorrect.

Figure 10 - View project details page (student)

Student Preferences Page

These are the pages where the student can view their preferences and change them.

A screenshot of a computer

AI-generated content may be incorrect.

Figure 11.1 - View preferences page

A screenshot of a computer

AI-generated content may be incorrect.

Figure 11.2 – Set or edit preferences page

Student Give Peer Review Page

Allows students to leave a peer review rating and comment o groups members of chosen project

A screenshot of a computer

AI-generated content may be incorrect.

Figure 12 - Give peer review page

Convener Homepage

This is the convenor's homepage where they are taken after logging in. It provides a list of projects they are currently managing and a navigation bar to navigate the website.

A screenshot of a computer

AI-generated content may be incorrect.

Figure 13 - Convenor home page

Convenor Create Project Page

This is where the convenor can create a new project by filling in the relevant details. The edit page is the same, except it updates projects instead of creating a new one.

A screenshot of a computer

AI-generated content may be incorrect.

Figure 14 - Convenor create project page

Convenor Assign Students Page

This is where the convenor can assign students to projects

A screenshot of a computer

AI-generated content may be incorrect.

Figure 15 - Assign students to project page

Convenor Allocate groups page

A screenshot of a computer

AI-generated content may be incorrect.This is where the convenor can auto-allocate students into groups based on preferences

Figure 16 - Group allocation page

Convenor View Project Pages

Shows a list of projects they manage and the details of the project like groups and students assigned

A screenshot of a computer

AI-generated content may be incorrect.

Figure 17 - Convenor list of projects page

A screenshot of a computer

AI-generated content may be incorrect.

Figure 18 - Convenor view project details page

Convenor edit Project page

Lets the convenor delete projects or edit details such as project title, description, deadline, and Group capacity.

A screenshot of a computer

AI-generated content may be incorrect.

Figure 19 - Convenor edit project page

Convenor View and Set Marks for Submissions

Lets convenor see a list of submissions for a project and return marks.

A screenshot of a computer

AI-generated content may be incorrect.

Figure 20 - View submissions and return mark page

Convenor View Peer Review and Adjust Marks page

Lets the convenor look at peer reviews for a group on a project and see the adjusted marks after mark distribution. Lets them override the adjusted marks.

A screenshot of a computer

AI-generated content may be incorrect.

Figure 21 - Convenor view peer review and adjusted marks

**4.9 Security**

Security was an essential requirement I wanted to meet as the project will be handling a lot of sensitive data, such as student data, peer reviews and marks. To implement this, I used Spring Security to handle the authentication and access control across the application[15].

When users register or are created in the system, their passwords are securely stored using BCrypt hashing. This ensures that even if the database is compromised, plaintext passwords are not exposed. The application does not store or transmit raw passwords at any point.

Restrictions have been placed on certain areas of the application based on the role of the user. When a user registers, they are assigned a role (convenor or student), which is checked before granting access to pages. For example, only convenors have access to group management as well as the ability to see all the peer reviews left for a project.

The system enforces authentication before accessing any route. Users who attempt to visit any protected endpoint without being logged in are automatically redirected to the login page.

Below is a simplified version of the security configuration class that sets up this behaviour.

The full version can be found in Appendix A – SecurityConfig.java

A computer code with many letters and numbers

AI-generated content may be incorrect.

Figure 22 - Simplified Security Configuration Class

# Chapter 5

# Testing and Evaluation

In this chapter, I will review all the testing done on the project. It will talk about the tests carried out, the results, any bug fixes I had to make, and the overall evaluation of the testing.

## 5.1 Overview of Testing Approach

To test the system, I used both manual testing and some automated tests to ensure that the core functionality is working as expected. I first carried out manual testing to check that all the major actions on the frontend of the system were working. This includes registering and logging in, submitting preferences, creating projects, allocating groups, etc. These tests helped discover any mistakes or bugs and resolve them.

Along with manual testing, I also carried out a few unit tests using JUnit. These tests included testing the compatibility score calculation between two students, checking the average compatibility score calculated for a group, and verifying that the group allocation algorithm produced balanced groups when run on small test datasets. To ensure security, I also tested to see if the raw password was stored after being encrypted, not just as plain text.

## 5.2 Manual Testing

Table 2 - Manual test and results

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case** | **Expected Result** | **Actual Result** | **Notes** |
| Student Registration | Student is registered and saved to the database | Student successfully created and saved | Works as intended |
| Student Login | Valid login redirects to student home page | Redirect successful | User credentials matched |
| Submit Preferences | Preferences saved and linked to student | Preferences saved | Tested with multiple options |
| Update Preferences | Set preferences can be seen and changed | Updates preferences | Student can see and update preferences once set |
| Convenor Registration | Convenor is registered and saved to database | Convenor successfully created and saved | Works as intended |
| Convenor Login | Convenor can log in with correct credentials | Login successful | Redirected to convenor dashboard |
| Project Creation | Project is saved with correct details | Project visible in project list | Checked all fields |
| Add Student to Project | Student appears in project | Student-project relationship saved | Verified in DB |
| View Students in Project | Convenor sees list of assigned students | All students displayed correctly | Shows list of all assigned students |
| Initial Group Allocation | Allocates based on preferences | Balanced groups created | Creates groups with good compatibility |
| View Groups | Students can see group info and members | Members and project title displayed | Correct for all test groups |
| Peer Review Submission | Students can review group members | Scores and comments saved | Values persisted in DB |
| Peer Review Access Control | Students cannot review non-group members | Access denied | Properly secured |
| View Peer Review After Deadline | Students see message and previous review | Edit blocked after cut-off | Message shown correctly |
| Convenor Views Peer Reviews | Convenor can see scores and comments | Accurate reviews shown | Sorted by reviewer |
| Mark Distribution Calculation | Final mark adjusted using clamp range | Mark adjusted within 0.9 to 1.1 | Max 100 enforced |
| View Adjusted Marks | Convenor sees adjusted mark per student | Marks shown in peer review view | Clamp logic verified |
| Submission Mark Entry | Convenor inputs group submission mark | Stored and used in mark distribution | Accepts valid numeric input |
| Group Deletion | Removes group and memberships | Group and related links deleted | No orphan records |
| Submission Access | Students see submission section after group is formed | Section shown based on group | Shows up after groups are allocated |
| Prevent Unauthenticated Access | Users not logged in are redirected to login page | Login page shown | Tested all protected routes |
| Role-Based Access | Convenor pages inaccessible to students and vice versa | Access denied | Routes secured correctly |

While conducting manual testing, I came across the access control issue where students were able to just type in the URL of the convenors' homepage and have access to features such as creating a project or viewing a list of all students. This is because after redirecting them to the right home page, I never implemented any restrictions on accessing certain web pages. To solve this issue, I added access control in my security configurations file, where only users with the role Convenor can access convenor pages and vice versa.

All major functional requirements were met after some minor bugs were fixed.

## 5.3 Unit Testing

The first focus of my testing was to make sure that the compatibility between the students was calculated correctly using their set preferences. To do this, I set up two students with known preferences and calculated their compatibility score manually. Then I put them through the calculateCompatibility() method to see if it gave me the correct output. Initially, I discovered that there was an issue with how the teamwork experience and communication skills were being calculated. Instead of pairing students to ensure there is a balance between them in terms of those skills, it awarded those with high communication and teamwork skills while penalising those with low-level skills. To fix this, I changed the logic so that the combined skill level was compared to an ideal value of 6, where being closer to it gave a higher compatibility score. After these changes, the test passed. (Result in Appendix B.1)

The second test was to ensure the method evaluateGroupCompatibility() was working as intended. This was an important test as it will be what I use to check if the algorithm is grouping the students correctly. To test this, I created and calculated the compatibility between 3 students. I used the calculateCompatibility() method between all the possible pairings in the group and averaged their score. This score was then compared to the score given by the method, and it passed the test. (Result in Appendix B.2)

The third test was to check if the password was encrypted correctly before being stored in the database. First, I set up a raw password, which was then used to register a student manually. Once the student had been saved into the database, I retrieved the password from it and compared it to the raw password to make sure they don’t match. Then I decrypted the hashed password and compared it to the raw password to check if it matched, and the password was the same after encryption. The test passed. (Result in Appendix B.3)

I also added another unit test, which randomly allocates groups and gets an average of the group compatibility score. After 10 runs, it prints out the average compatibility score of each run. This is used later to test if the allocation algorithm is working effectively.

## 5.4 Testing Group Allocation Algorithm

The evaluateGroupCompatibility method is used to calculate how well a group of students has been formed based on individual compatibility scores. It goes through each group in a given project, retrieves all members of that group, and then works out a compatibility score for every unique student pair within the group. These individual scores are calculated using the calculateCompatibility method. If either student in a pair is null or something goes wrong during the comparison, a neutral fallback score of 0.5 is applied. After looping through all valid pairs, the method calculates the average compatibility score for the group, which reflects how well the group has been matched based on preferences.

### 5.4.1 Methodology

To test the integrity and consistency of the group allocation algorithm, I decided to run multiple allocations on the same project and use the evaluateGroupCompatibility method to score the results. For each run, I cleared any existing groups and reallocated students using the same preference data. Once the groups were generated, I used the method above to calculate the average compatibility score for every group in that allocation, and then worked out the overall average across all groups. This process was repeated ten times in total to see how much variation there was between each allocation. Evaluating heuristics across diverse problem instances is essential, as no single heuristic outperforms others universally [17]. The goal was to observe whether the algorithm consistently formed reasonably compatible groups or if the results varied significantly between runs.

### 5.4.2 Results

The group allocation was run 10 times on a project with 31 students (all with preferences) and a group capacity of 4.

The first round of allocation yielded these results (Testing data in Appendix C.1):

* Group 100 (Score: 0.66)
* Group 101 (Score: 0.76)
* Group 102 (Score: 0.70)
* Group 103 (Score: 0.70)
* Group 104 (Score: 0.67)
* Group 105 (Score: 0.72)
* Group 106 (Score: 0.64)
* Group 107 (Score: 0.67)

To calculate the average score, we sum all the group scores and divide by the number of groups:

Score = 0.66+0.76+0.70+0.70+0.67+0.72+0.64+0.67 = 5.52

The students were split across 8 groups, with the group compatibility score varying from 0.64 to 0.76. The average score calculated out to be 0.69. I repeated this process 10 and the scores for each allocation are as follows:

* **Run 1 - 0.69**
* **Run 2 - 0.681**
* **Run 3 - 0.67**
* **Run 4 - 0.691**
* **Run 5 - 0.665**
* **Run 6 - 0.665**
* **Run 7 - 0.688**
* **Run 8 - 0.702**
* **Run 9 - 0.67**
* **Run 10 - 0.696**

The graph below illustrates how the average scores only slightly vary between 0.665 and 0.702, showing that the results are generally consistent across all runs. This implies that the algorithm generates groupings in terms of compatibility that are stable and consistent.

A graph with a red line

AI-generated content may be incorrect.

Figure 23 - A line graph showing the average group compatibility score across 10 allocations

### 5.4.2 Comparing results to random allocations

To verify that the algorithm is effective and is doing what it is meant to be, I choose to compare the results of the allocation to randomly allocated groups. I did this by creating a test in GroupServiceTest where it would randomly allocate groups for the same project with the same students and return the average group compatibility over 10 runs. The results were (Available in Appendix C.2):

* **Run 1: 0.589**
* **Run 2: 0.578**
* **Run 3: 0.605**
* **Run 4: 0.609**
* **Run 5: 0.591**
* **Run 6: 0.589**
* **Run 7: 0.602**
* **Run 8: 0.621**
* **Run 9: 0.598**
* **Run 10: 0.592**

The average of these 10 runs was 0.59. This is 0.10 lower than the average of the one given by the algorithm, which was 0.69. This indicates a relative improvement of 16.95%, which is a solid increase over random allocation. In addition to this, random allocation doesn’t take into consideration constraints such as balancing out groups evenly to ensure there is a fair number of students among groups, avoiding grouping together students who prefer leading, etc. The overall range and average of the algorithm are higher, as shown in Figure 24 below, showing it is carrying out its intended purpose.

A graph with a red line and blue line

AI-generated content may be incorrect.

Figure 24 - Line graph showing average algorithm vs random allocation group compatibility score

# Chapter 6

# Critical Appraisal

## 6.1 Critical Summary

This project set out to create a web-based tool for managing student software projects, with a particular focus on automating the group allocation process, implementing peer review functionality, and enabling fairer distribution of project marks. Based on the aims and requirements outlined at the start, I believe the project has met its goals successfully.

The group allocation algorithm, which assigns students into groups according to compatibility scores based on personal preferences, was the system's main component. The average compatibility score across several allocations remained within a stable range, demonstrating that the algorithm could reliably produce reasonably compatible groupings. While not perfect, the groups produced were more balanced and preference-aware than what would typically occur with manual or random allocation, meeting the project’s goal of supporting convenors and improving fairness in group formation.

Another key aim was to provide a mechanism for students to peer review each other and for these reviews to feed into an adjusted mark distribution system. This was implemented with a normalisation-based formula using a clamped range of 0.9 to 1.1 to ensure fairness without over-penalising or over-rewarding students. The peer review system also had some restrictions put in place to make it more sensible, such as updates only being allowed for a short period after the project deadline. Together, these mechanisms improve transparency and provide a way to account for differences in contributions during the project.

Throughout development, the system remained true to its MVC architecture. Controllers remained thin, services handled core logic, and repositories handled data access. The system was simpler to expand and maintain as a result of this separation. Some of the features which made sure that the system was not only functional but also robust and secure included secure authentication, role-based access for students and convenors, and strict validation of forms.

With all that in mind, there undoubtedly exists room for continued optimisation. While current heuristic matching works incredibly well, I could implement clustering algorithms such as K-means to identify more intelligent clusters, especially in bigger data sets where preference overlap grows increasingly complex. Similarly, compatibility scoring can also be extended to include factors such as balance of workload, academic achievement, or history of working together. These would advance the system closer to real-world expectations and institutional needs. To improve it further, the UI could be customised more by adding a more modern and responsive layout. Although it functions as needed, an improvement in the UI will improve the overall user experience.

In addition to this, I could consider the possibility of hosting the system online using platforms such as Railway or Render. With my infrastructure in place, setting up a way to host could be seamless. Alongside this, I could implement a file upload and download system to let convenors upload more intricate and detailed project information, as well as allowing students to upload their projects with their submission. These files can be stored on a cloud service like AWS S3, with the file path being stored in the database so the files can be easily retrieved. By doing this, the system would become more comprehensive and independent, eliminating the need to use third-party learning platforms like Blackboard.

## 6.2 Discussion of Context

This project was designed with the higher academic context in mind to make group-based software projects more manageable for students as well as convenors. It solves a common pain in higher education—fair, yet efficient group allocation—through an automated system that works in conjunction with student preferences to create moderately compatible groupings. Even though project management software exists in industry, few have the specific use case of academic modules with allocations, peer review, as well as fair-based allocation of marks.

The platform, though built for university environments, could be adapted to other applications involving team creation, such as employee project allocation or even hackathon team creation, where participants' preferences or soft skills are collected. The algorithm used in group assignments, as well as features such as peer review and adjusted marking, could provide an even clearer and explainable way of tracking contributions in collaborative working spaces.

Since the project was designed using Java Spring Boot and a standard MVC format, it is fairly easy to continue maintaining or further extend. For example, if some departments want to use the system, an enhancement in the future can be to adopt departmental project pools and access scopes for users. Platforms like Railway or Render could facilitate hosting this to be online, and cloud storage facilities like AWS S3 could facilitate features like file uploading. All these additions would take the tool from a prototype to a complete, production-ready system.

In addition, the system promotes sustainability in education by reducing the academic staff workload, making previously labour-intensive processes like group assignments and moderation of performance automatic. It provides an evidence-based solution that guarantees transparency in decision-making, a valuable feature in the academic community. With an enhanced UI and more configurability to meet project-based requirements, this tool is usable in the long term in real educational environments.

## 6.3 Personal Development

Working on this project has pushed me to develop both technically and personally. I’ve improved a lot in areas like backend development with Spring Boot, designing systems that are modular and following the best practices like the MVC model, and just generally thinking more like a software engineer rather than just a student writing code to get things working. I also learned more about managing real-world complexity—whether that was choosing how to write the group allocation logic so it was equitable, or choosing how to link all the different features like peer review and marking together without breaking things.

This project also made me more independent when debugging and troubleshooting by myself, especially when doing something like database relationships, Spring Security, or trying to keep users from ruining the system in weird ways. I've become more confident in writing neat code and testing it well, and I have learnt the importance of time management, keeping development and dissertation writing aligned with other university deadlines. In general, I believe I've grown substantially, and now I know what it takes to create a full-sized software project from backend to frontend.

# Chapter 7

# Conclusion

This project successfully met its overall aim of creating a tool to manage student software projects, focusing on fair and preference-based group allocation, submission management, peer review, and adjusted mark distribution. All the key functional and non-functional requirements outlined at the start were implemented and tested. The system follows the MVC model and integrates Spring Boot, Thymeleaf, MySQL, and Spring Security to provide a secure and structured backend.

The group assignment algorithm consistently generated relatively compatible groups based on student preference, and testing showed consistent mean compatibility scores across many test cycles. Additional features such as peer review, adjusted marking, and submission tracking of projects were integrated to reflect actual academic procedures.

The project also laid a good groundwork for future improvements, including smarter allocation algorithms (e.g., K-means clustering), improved UI, and optionally file storage integration through cloud services. In general, the system can be used, is modular, and nearly ready for real-world deployment with little additional effort.

# References

[1] D. Gale and L. S. Shapley, “College admissions and the stability of marriage,” American Mathematical Monthly, vol. 69, no. 1, pp. 9–15, 1962.

[2] D. J. Abraham, R. W. Irving and D. F. Manlove, “The student-project allocation problem,” in ISAAC 2003: Algorithms and Computation, Kyoto, Japan, 2003.

[3] H. W. Kuhn, “The Hungarian method for the assignment problem,” Naval Research Logistics Quarterly, vol. 2, no. 1–2, pp. 83–97, 1955.

[4] A. K. Jain, “Data clustering: 50 years beyond K-means,” Pattern Recognition Letters, vol. 31, no. 8, pp. 651–666, 2010.

[5] L. E. Gueldenzoph and G. L. May, “Collaborative peer evaluation: Best practices for group member assessments,” Business Communication Quarterly, vol. 65, no. 1, pp. 9–25, 2002.

[6] P. Fleckney, J. Thompson and P. Vaz-Serra, “Designing effective peer assessment processes in higher education: a systematic review,” Higher Education Research & Development, vol. 43, no. 1, 2024. [Online]. Available: https://doi.org/10.1080/07294360.2023.2235597

[7] P. A. Ertmer et al., “Using peer feedback to enhance the quality of student online postings: An exploratory study,” Journal of Computer-Mediated Communication, vol. 12, no. 2, pp. 412–433, 2007. [Online]. Available: https://doi.org/10.1111/j.1083-6101.2007.00331.x

[8] P. R. K. Jones et al., “Using peer review to distribute group work marks equitably between medical students,” Medical Education, vol. 51, no. 4, pp. 380–390, 2017. [Online]. Available: https://doi.org/10.1111/medu.13226

[9] J. E. Brindley, C. Walti and L. M. Blaschke, “Creating effective collaborative learning groups in an online environment,” The International Review of Research in Open and Distributed Learning, vol. 10, no. 3, 2009. [Online]. Available: https://doi.org/10.19173/irrodl.v10i3.675

[10] P. M. Papadopoulos and T. D. Lagkas, “Group formation algorithms in collaborative learning environments: A review,” Education and Information Technologies, vol. 25, pp. 1607–1635, 2020. [Online]. Available: https://doi.org/10.1007/s10639-019-10055-9

[11] E. Sánchez and R. Ramírez, “Heuristic approaches for the group formation problem with equity considerations,” Applied Soft Computing, vol. 67, pp. 764–778, 2018. [Online]. Available: <https://doi.org/10.1016/j.asoc.2018.03.038>

[12] R. Kumar, “Spring Boot Architecture,” *GeeksforGeeks*, Oct. 11, 2022. [Online]. Available: <https://www.geeksforgeeks.org/spring-boot-architecture/> [Accessed: Apr. 30, 2025].

[13] *Spring Framework Documentation*, “Thymeleaf,” Spring.io. [Online]. Available: <https://docs.spring.io/spring-framework/reference/web/webmvc-view/mvc-thymeleaf.html>. [Accessed: Apr. 30, 2025].

[14] D. J. Barnes and M. Kölling, *Objects First with Java: A Practical Introduction Using BlueJ*, 6th ed., Pearson Education, 2016.

[15] Spring Security Reference Documentation, “Spring Security Architecture,” Spring.io. [Online]. Available: <https://docs.spring.io/spring-security/reference/servlet/architecture.html>

[16] S. A. Munson and R. Priedhorsky, “Maximizing Satisfaction in Group Formation,” University of Minnesota, 2007. [Online]. Available: <https://www.smunson.com/portfolio/projects/groupform/paper.pdf>[smunson.com](https://www.smunson.com/portfolio/projects/groupform/paper.pdf?utm_source=chatgpt.com)

[17] I. Dunning, S. Gupta, and J. Silberholz, “What Works Best When? A Systematic Evaluation of Heuristics for Max-Cut and QUBO,” *Informs Journal on Computing*, 2015. [Online]. Available: <https://www-personal.umich.edu/~josilber/SECM_clean.pdf>

# Appendices

## Appendix A – SecurityConfig.java

@Configuration  
@EnableWebSecurity  
public class SecurityConfig{  
  
 @Autowired  
 private AuthenticationHandler authenticationSuccessHandler;  
  
 @Bean  
 public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {  
 http  
 .authorizeHttpRequests((requests) -> requests  
 //allow login page and static files  
 .requestMatchers("/home","/login","/register", "/images/\*\*", "/css/\*\*").permitAll()  
 .requestMatchers("/convenor/\*\*").hasRole("CONVENOR")  
 .requestMatchers("/student/\*\*").hasRole("STUDENT")  
 //everything else requires login  
 .anyRequest().authenticated()  
 )  
 .formLogin((form) -> form  
  
 //Tells spring to use custom login page  
 .loginPage("/login")  
 .successHandler(authenticationSuccessHandler)  
 .permitAll()  
  
 )  
 .logout((logout) -> logout  
 .logoutSuccessUrl("/login?logout")  
 .permitAll()  
 )  
 .exceptionHandling((exceptions) -> exceptions  
 .accessDeniedHandler((request, response, accessDeniedException) -> {  
 if (request.isUserInRole("CONVENOR")) {  
 response.sendRedirect("/convenor/home?error=accessDenied");  
 } else if (request.isUserInRole("STUDENT")) {  
 response.sendRedirect("/student/home?error=accessDenied");  
 } else {  
 response.sendRedirect("/login?error=accessDenied");  
 }  
 })  
 )  
 .csrf(AbstractHttpConfigurer::disable);  
  
  
 return http.build();  
  
 }  
  
  
 //Used to encrypt the password  
 @Bean  
 public PasswordEncoder passwordEncoder() {  
 return new BCryptPasswordEncoder();  
 }  
  
}

## Appendix B – Unit testing results

### Appendix B.1 – Password testing results

**A screen shot of a computer

AI-generated content may be incorrect.**

### Appendix B.2 – Student compatibility testing results

**A screenshot of a computer

AI-generated content may be incorrect.**

### A screenshot of a computer AI-generated content may be incorrect.Appendix B.3 – Group compatibility testing results

## Appendix C – Group Allocation Algorithm tests

### Appendix C.1 – Allocation algorithm testing data

**A screenshot of a computer

AI-generated content may be incorrect.**

**A screenshot of a computer

AI-generated content may be incorrect.**

**A screenshot of a test

AI-generated content may be incorrect.**

**A screenshot of a computer

AI-generated content may be incorrect.**

**A screenshot of a group

AI-generated content may be incorrect.**

**A screenshot of a white sheet

AI-generated content may be incorrect.**

**A screenshot of a group

AI-generated content may be incorrect.**

A screenshot of a test

AI-generated content may be incorrect.

A screenshot of a group

AI-generated content may be incorrect.

A screenshot of a test

AI-generated content may be incorrect.

### Appendix C.2 – Average group compatibility score of random allocation

**A screenshot of a computer

AI-generated content may be incorrect.**