



UNIVERSITY OF
LEICESTER

**Department of Informatics
University of Leicester
CO7201 Individual Project**

Final Report

Web Application for Allocating Groups,
Topics, and Supervisors for Group
Discussions

Ashwini Kabbali Channabasavadevaru

akc17@student.le.ac.uk

229027705

Project Supervisor: Airantiz,Dimitrios

Second Marker: Dr Paula Severi

Word Count: 10,538

8 /09/2023

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 amount to plagiarism and will be considered grounds for failure in this module and the degree examination as a whole.

Name: Ashwini Kabballi Channabasavadevaru

Date: 8/09/2023

Abstract

In the realm of University Education, forming a group with like-minded students can be a challenging task. This involves a lot of effort in finding students who have similar interests. In current applications, where students are required to select their own group members, it often results in a diverse group of students with varying interests which might lead to conflicts between students and dissatisfaction among the group members. When any conflict arises, it requires the administrative staff to intervene to reallocate the groups, change topics or supervisors. As this requires human intervention, this might not provide an optimal solution. To solve this given problem, a web application that allocates students to groups automatically based on the student preferences is implemented. This allocates the students to groups, topics, and supervisors automatically by saving time and effort.

In our application, the Supervisor uploads the topic to the system and the students choose their four preferences according to their most preferred topic to their least preferred topic in order of their priority from the given set of topics. A matching algorithm that allocates students to groups based on their preferences is implemented. This also prioritizes students who have submitted their preferences on a first-come first-served basis. This will produce a group of students who have a similar interest and allocate the supervisors to each group. The admin sets the group size and initiates the algorithm, which then allocates students to the given group size. This application eliminates the need for the intervention of university staff, which helps in saving time and manual effort.

Table of Contents

1	Introduction	5
1.1	Aim	5
1.2	Challenges	5
1.3	Risks	6
1.4	Background Research and Literature Review	6
2	Requirements.....	7
2.1	High-level Requirements	7
2.1.1	User login authentication, roles, and permission	7
2.1.2	Topics Preferences and Submission	8
2.1.3	Algorithm Implementation	8
2.2	Detailed Requirements	8
2.2.1	Recommended Requirements	9
2.2.2	Optional Requirements	9
3	Technical Specification	10
4	Methodology	12
4.1	Software Development Approach.....	12
4.2	Client-Server Architecture	12
4.3	Milestones.....	13
5	Design and Architecture	15
5.1	System Architecture Design.....	15
5.2	Use Case Diagram	16
5.3	Database Design and Schema	18
5.4	Algorithm Design and Analysis	20
6	Project Outcome	21
6.1	Client-Side Modules	21
6.1.1	App Components.....	21
6.2	Server-Side Modules	27
6.3	Code Snippets.....	31
6.4	Libraries and Plugins	35
7	Software Testing	36
8	Conclusion.....	38
8.1	Challenges Faced	38
8.2	Feedback	39
8.3	Future Improvement.....	39
9	References	41

1 Introduction

1.1 Aim

The main goal of this project is to create a web application that will allocate students to groups, topics, and supervisors to the Personal and Group Skills Module (CO7210). This application involves three users named - Admin, Supervisor and Student. In which, Supervisors can specify the topics which they are willing to supervise. Supervisors can add or edit the topics which they are willing to supervise. Students have the option of selecting their preferred topics, they can select four topics minimum in order of their preferences from a list of given topics. Once the Student submits their topic preferences, the web application will store the preferences, and the allocation of students to groups, topics and supervisors is done automatically after the deadline. This allocation is achieved by implementing the matching algorithm, where each student is preferably matched to their first topic, then the second and so on, which is discussed thoroughly in the later part of the report. Each group consists of 4 to 5 students who have the same topic. The database is designed to store a list of topics and preferences made by each student. The admin can view the overall process of the application which involves viewing and updating the preferences set by the students, and see the output of the algorithm, if the admin agrees with the output produced by the algorithm, the results are published, or an email is sent consisting of the information of the group allocation. Once the allocation is done, students will receive an email with the topics, names, and email addresses of the members of the same group. The published topics are made visible to the students when they login to their systems. Supervisors can view the student groups that they will be supervising once the results are published.

1.2 Challenges

- The Author is not well versed with the framework: React to implement the front-end with Bootstrap, and Express.JS for the back-end implementation. To learn and understand the new framework is challenging.
- The Sequelize ORM used is new to the Author, this takes a lot of research and documentation to be ready to work on it. Using Sequelize ORM to connect Models was a challenging task, as it involved a lot of research to understand the connection between the models.
- Finding the best matching algorithm is challenging as it involves going through all matching algorithms and comparing their advantages and disadvantages before using it in the application to best suit the development. Many research papers were referred by the Author to find the best matching algorithm that is suitable for building the application.
- While testing the application, many errors were discovered, which required fixing them. This process consumes more time as there are interdependencies between various services.

1.3 Risks

- The author is new to the React framework and Sequelize ORM, hence learning throughout the development of the web application is required to build a successful application.
- Implementing all the features of this application can be difficult due to the limited time window to complete this project. A lot of third-party libraries need to be learned before using them and need to check the compatibility of the APIs with the main application needs to be addressed.
- The algorithm needs to be designed for efficiency and correctness. If the algorithm isn't producing desired results the whole application fails to meet its purpose.
- The integration of the front-end and the back-end can be complex. This needs to be thoroughly tested for the proper working of the application.
- Testing each user scenario for positive and negative scenarios consumes time and the application requires thorough testing for all scenarios to produce accurate results.

1.4 Background Research and Literature Review

The application is developed using Express.js which is a web application framework for Node.js. Express.js is one of the popular frameworks available to build RESTful APIs (Application Programming interfaces). One of the key features of this framework is the Middleware function. This is one of the best frameworks for building API endpoints to communicate with the application front-end using HTTP protocol methods. Node.js is the server-side runtime environment which is used to run server-side JavaScript code. Node.js is highly recommended since it is suitable for building real-time web applications. To write a database, Sequelize Object-Relational Mapping is used as it helps in interacting with the database efficiently. This is used in relation to MySQL database, which is a popular database to store and retrieve data for carrying out CRUD (Create, Read, Update and Delete) operations. JWT token is used to authenticate the token. The React library is used in the front end to build user-friendly web applications.

To gain more knowledge on the requirement of this application, which is to implement the matching algorithm, many research papers have been considered, which explain how to overcome the lack in the current system of allocating students to groups.

Allocation systems are essential in educational institutions to efficiently allocate students to courses, programs, or housing. The distribution of students has traditionally been done via manual processes, lottery-based systems, and first-come, first-served strategies. However, these approaches frequently have problems with efficiency, fairness, and scalability.[1]

To overcome this issue, an automated allocation system was introduced. Students peruse project descriptions and create lists of their choices, while supervisors have their own preferences for both students and projects. A few strategies have been implemented to provide a consistent allocation mechanism, but no one technique has received widespread support. Researchers have suggested new SPA models and enhanced stability indicators. A three-phase approach has also been recommended as a time-saving measure to cut down on tasks that are assigned at random. [1]

Methodology for project allocation is used by GC-UESTC, a program where students graduate from both UESTC and the University of Glasgow. Students have a few weeks to review the projects and get in touch with supervisors for explanations once staff members from both universities post them to a project database.[2]

Students then rank Glasgow and UESTC staff members according to their preferred projects, individually. Projects are assigned based on a variety of factors, including the intended load distribution and the number of projects a staff person may manage, using a matching algorithm.

The algorithm seeks to minimize mismatches or maximize student preferences. When the algorithm cannot assign a project, the remaining students are chosen manually. This method seems fair, ensuring that student preferences are considered, and saves time. [3]

However, the suggestion advises using a modest proportion of pre-allocations to alleviate this problem because it does not ensure students or staff find their desired options. In addition to the allocation methods mentioned earlier, many higher education institutions employ alternative approaches. They can either select a project from a list of titles proposed by supervisors or propose their own project titles. Moreover, several schools and departments, especially those outside the U.K., prefer final-year projects to be conducted in groups.[4][5]

This group project approach serves multiple purposes. It not only reduces the workload for supervisors, making the allocation process more straightforward, but it is also aimed at fostering teamwork, enhancing communication, and developing leadership skills among students during their final-year projects.

Author Anwar proposed an alternative tactic to address this issue. With this modified method, individuals continue ordinarily prioritizing their project preferences independently, yet additionally attain the prospective to cooperatively organize committees of up to three partakers with compatriots who similarly aspire for a shared undertaking selection.[6]

By allocating projects across diverse topics, this approach fosters collaborative learning through valuable opportunities for students to jointly work on pursuits that mutually pique their interests.

2 Requirements

2.1 High-level Requirements

2.1.1 User login authentication, roles, and permission

- User Registration and Login functionality with verification of users.
- User authentication is done based on JWT tokens.
- Users are divided into three user types based on their roles:
 - Admin.
 - Supervisor.
 - Student.
- User Permissions are given based on the roles specified.
 - Admin has full permission to access the application.
 - Permissions are restricted to users based on the user type.

2.1.2 Topics Preferences and Submission

- Supervisors can add, edit, or delete the topics that they are willing to supervise.
- Students can view all the topics that are available and the corresponding supervisor's names.
- Students should select their 4 preferred topics from a list of topics available.
- Students are made to select the topics in order of their preferences from the most preferred to least preferred.
- Students are given permission to change their topic selection as many times as they like till the deadline is over.
- A deadline is set for the students to submit their preferences.

2.1.3 Algorithm Implementation

The matching algorithm is implemented to allocate students to groups, students and supervisors based on the topics submitted by the students.

Each group will have students consisting of 4 or 5 members with the supervisors allocated to the topics. Students are allocated to the group mostly based on their first preferences, then second and so on. The algorithm should produce a group of students whose preferences are the same. The algorithm should also assign groups to those students who haven't set their preferences. The group produced by the algorithm should make sure groups for the submitted topic preferences are assigned and the allocated group should have unique topics assigned to each group. Algorithms like the Hungarian maximum matching algorithm, Stable marriage algorithm, and Weighted Bipartite algorithms have been considered to decide on the algorithm. The author has decided to implement the algorithm using the Stable Marriage Algorithm which is the suitable algorithm approach to design the algorithm to allocate students to groups.[7][8][9]

2.2 Detailed Requirements

Administrator Roles:

- Admin is given permission to view, edit or delete topics added by the Supervisor.
- Admin can view the list of all available Supervisors and Students. Admin can add, edit or delete Supervisors' and Students' details.
- Admin is given permission to view and edit preferences submitted by the students.
- Admin has the authority to set group sizes and run the matching algorithm to form groups.
- Admin can view the output of the matching algorithm, if the admin is satisfied with the groups resulting from the algorithm, he/she can publish the allocation of topics, groups and supervisors to the students and make it visible to supervisors as well.
- Admin can make changes to the groups and topics whenever necessary.
- Admin can filter Students and Supervisors.
- Admin can notify the students and Supervisors about group allocation by sending an email with their allocated groups, names, and email addresses of the students of the same group and the corresponding Supervisor.

Supervisor Roles:

- The supervisor is given permission to add, edit or delete the topics they are willing to supervise.
- Supervisors can see the results of the student groups allocated to them.

Student Roles:

- Once the Student logs into the system, a list of topics with the corresponding supervisor's name is made available.
- Students are provided with a preference page to select 4 topics from a list of topics shown based on their order of preferences from most liked to least liked topics.
- Students are given permission to make changes to their topic selection as many times as they want till the submission deadline date.
- Students can view their allocation of groups, topics, and supervisors once the admin publishes the results.

2.2.1 Recommended Requirements

- Admin can make announcements whenever there is a need to convey information. Announcements which are announced can be edited or deleted by admin. The announcements are displayed in which the top column consists of the latest announcement.
- The announcement that is made by admin is visible to Supervisors and Students on their respective Homepages under the announcement heading.
- Admin should be able to set deadlines for the Project topic Preference selection and should not allow any students to set preferences after this deadline, this will help in completing the allocation process in a timely manner and helps the students to work on their topics as soon as possible.
- The System should allow students to change their preferences any number of times till the deadline is over. This will allow students to be flexible in their preferences and if students change their minds about their preferences, they should be able to change their preferences without any penalty.

2.2.2 Optional Requirements

- In addition to all the features mentioned above the web application could be extended to have, a set deadline feature that will help the institution to have a specific time to receive all students' preferences and have a deadline to disable Supervisors to upload topics so that there won't be any changes made once the admin runs the algorithm to allocate students to groups.
- The web application could be extended to form a discussion forum for group members to communicate with each other.

3 Technical Specification

The Technical Specification that is required to build this web application are listed in Table 1, after a thorough research on what is best to build this web application. The front-end is built using React, the backend is developed using Express.js which is integrated with Node.js which is combined with MySQL database.

Front-end technology.

- ReactJS – React is a JavaScript library that is used to build the user interfaces of the applications. [15]
- HTML5 and CSS3- Languages used to structure and design the application's visual presentation. HTML is a markup language used to create the structure of web pages. CSS is used to style web Pages. CSS is used in combination with Bootstrap to style the application. [16][17]
- Material UI - This is an open-source React component library that implements Google's Material design. [21]
- Bootstrap - This is an open-source front-end framework that provides pre-designed components that helps in building responsive and visually appealing web application.[24]

Back-end technology:

- Node.js – It is an open-source, cross-platform runtime environment for server-side programming. [14]
- Express.js – It is a Node.js web application framework which provides features for building web applications.[19]
- MySQL – A relational database, SQL is used for managing and manipulating data held in the relational database management system (RDBMS). [18]
- Sequelize – It is a Node.js ORM (Object-relational mapping) tool to interact with MySQL. This is a modern Typescript and Node.js ORM for MySQL server. [22]

API Development and testing:

- Postman – For testing the APIs of the development application Postman is used. This provides a user-friendly interface for sending HTTP requests. [20]

Version Control:

- SVN – Subversion is a version control system used to manage project source code and project files.

IDE:

- Visual Studio Code - This is a code editor used for building, testing, and debugging web and cloud applications. [23]

Table 1: Technical Specification

Components	Technology	Version
Programming Language & Tools.	JavaScript.	ES6.
	HTML & CSS	HTML5 & CSS3
	SQL	NA
Frameworks.	React	v18.2.0
	Express	v4.18.2
	Bootstrap	v5.3.1
	Material UI	v4.11.3
	Sequelize ORM	v6.32.1
Database Management Application.	MySQL	v8.0.30
Operating System	macOS	v13.5
IDE	Visual Studio Code	V1.7.4
Version Control	SVN	v1.14.2
Testing Tool	Postman	v10.11.1
Runtime Environment for Server-Side	Node.js	v18.12.0

4 Methodology

4.1 Software Development Approach

The approach carried out in the software development of this web application is Agile Methodology. In this methodology, Scrum Framework is followed, where software requirement specifications are given. Here, there are three main roles considered Product Owner, the Scrum Master, and the Scrum team. [13]

The Product Owner is the project proposal system that provides all the necessary requirements that are required for building the application successfully. This Project proposal system provides all the requirement specifications, deadlines and the final application that should be produced.

The Scrum Master who is the Supervisor conducts meetings to check upon the progress of the Project done by the Scrum team and provides feedback during each sprint. Each sprint will have all the important features that require development and testing and should be shown to the Supervisor to get feedback to check for any improvements and successful development of the application.[10]

The Scrum team being the author who is the developer and tester of this web application provides the deliverables in each sprint. The Scrum Master sees the work delivered in each sprint and provides feedback, based upon the given feedback, the changes are made to the application and shown in the next sprint to the Supervisor, this iterative process is carried out throughout the development of the complete application.

This approach to the development of applications ensures that there is continuous improvement in the development lifecycle and that the final application delivered aligns with the requirements effectively.

4.2 Client-Server Architecture

In the Client-Server architecture the functions of the system are divided between two components, the client, and the server, where the client sends the request to the server and the server responds to the request with the corresponding information.

Client-side is built using React.js along with Material UI, react-bootstrap, Modal and CSS to produce an aesthetically pleasing user-friendly interface. The client-side takes input from the users and sends a request to the server to get the user-required information. The users such as Admin, supervisors and Students will send request according to their given permissions and receives the information from the server which has information stored in the database. In this allocation of Students to group applications, the client provides an interface for the users to log in using the credentials, the dashboard to perform user actions such as publishing topics by the Supervisor, editing/deleting the topics by both the Supervisor and Admin, to set preferences by the students, to run the algorithm by Admin, to make announcements by Admin. The client collects all this user information and sends it to the server to process the information and produce appropriate results by sending the response to the client.

The server is the backbone of the application where all the logical components of the systems are handled. Server-side is built using the Express.js framework, Node.js runtime environment. The server handles all the application business logic. The server receives the client request, and

the server must handle user inputs by validating users and accessing the database using Sequelize ORM to retrieve information based on a particular request.

The controller component of the server has all the logic of the application, and middleware contains all the necessary permissions given to the users to access specific components of the application. Routes contain all the required routes to communicate with the application by performing Restful API requests. The server performs token validation by generating tokens using JWT token validation of users for authenticating and for the security of the application. The client sends a request using HTTP request to specific endpoints on the server, which process the request and sends it back to the client to display the results to the User. The client-server architecture advantage is that the client and server are both separated, the changes made to the client components or server components won't affect the other. This helps in the application being more scalable and efficient.

4.3 Milestones

The Milestones are written in below Table 2. The Gantt chart for the same is shown in below Fig.1 which shows the time plan.

Table 2: Milestones

Module	Task	Time
Requirement Gathering, Research on Application and Environment Setup.	<ul style="list-style-type: none"> Gathering all the requirements needed for the development of Application software. Research on all the tools and libraries required to build the application. Setting up the application by configuring the system with the required database, IDE, and testing platform Detailed plan for Project – Preliminary Report. 	Week 1-2
Database Design, API, and Server-Side development. (Registration, login, authentication)	<ul style="list-style-type: none"> The database models are designed to store details of users, topics, and preferences. APIs endpoints are written for testing the software functionalities with specific routes. Registration, login, and User authentication using JWT token is written. Admin credentials are taken care of in the backend. Supervisors and Students are provided with registration and validation of users is done. Users are logged in to the application and user verification is done using JWT token. 	Week 3-4

Client-side development and Server-side development.	<ul style="list-style-type: none"> Welcome page with Registration and Login. Registration and Login pages are created for Admin, Students and Supervisors. Server-side: Supervisors are given permission to add topics that they are willing to supervise. Supervisors, Students and Admin can view the topics added by supervisors. Admin and Supervisors are enabled with permission to delete the topics. Admin and Supervisors are given permission to edit the topics. 	Week 5-6
Client-side development and Server-side development.	<ul style="list-style-type: none"> Client-side: Supervisor Home Page, where admin's messages are displayed. Supervisor page to add topic, view topic, edit topic and delete topic. Admin Page to view topics uploaded by the supervisor, edit topic, and delete topic. Admin can view all the supervisors and students list with their names and email addresses. Students are provided with the option to set preferences and submit preferences. 	Week 7-8
Client-side development and Server-side development.	<ul style="list-style-type: none"> Admin is given the permission to add or edit user details as per Second Supervisor interim interview feedback. Admin is provided with the option to filter users upon second supervisor feedback. Users are provided with filter option to search topics upon second supervisor feedback. Algorithm to allocate students to groups. Admin is provided with option to send email to the corresponding supervisor and student. 	Week 9-10
Client-side development and Server-side development. Final report and testing the application.	<ul style="list-style-type: none"> Setting deadline and announcements. Final report writing and fixing all the errors in the application by testing the application for all positive and negative scenarios. 	Week 11-12

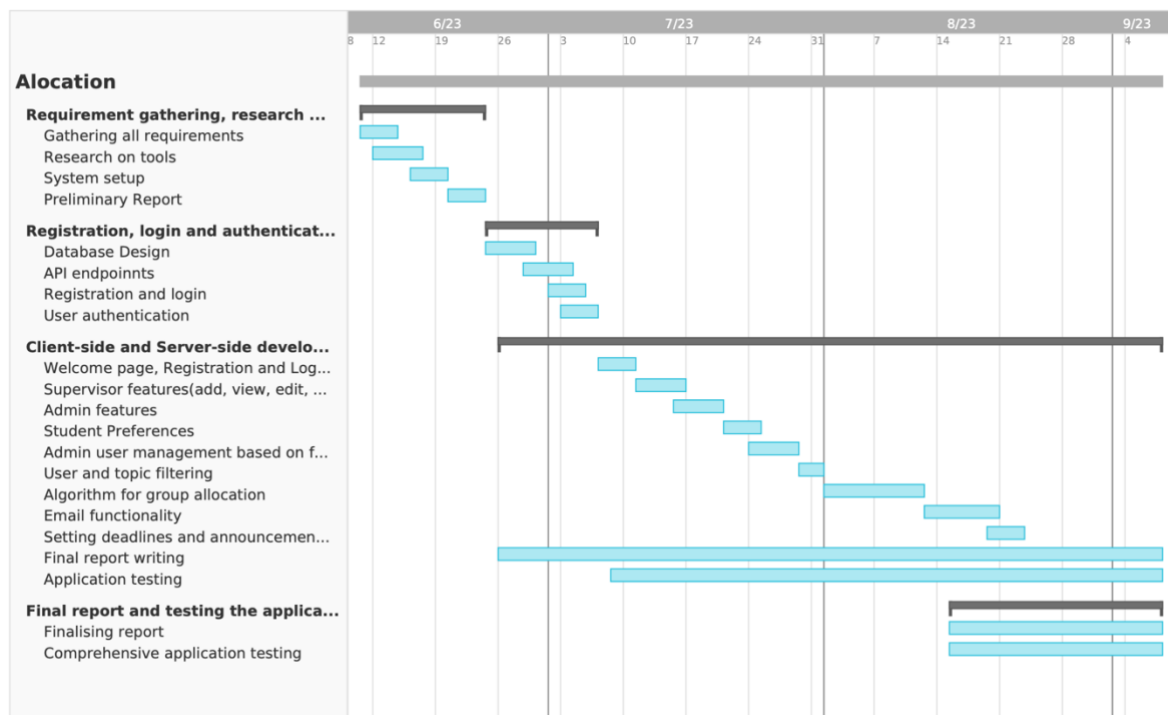


Figure 1: Gantt chart displaying time plan.

5 Design and Architecture

5.1 System Architecture Design

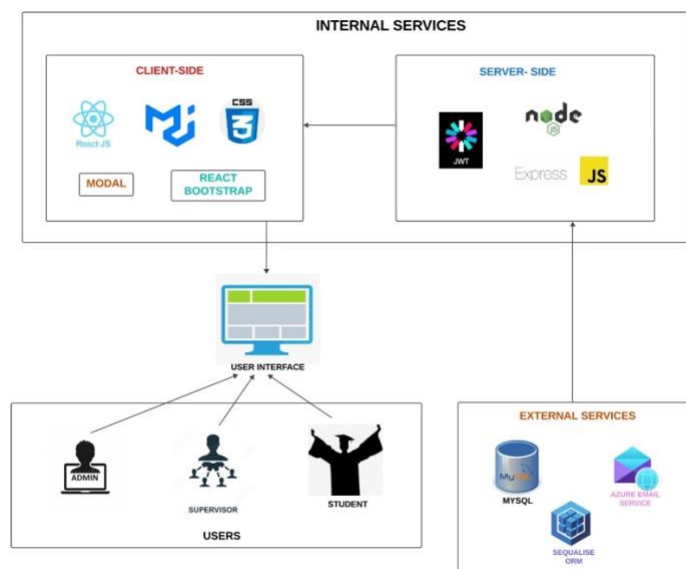


Figure 2: System Architecture

The System Architecture of the Application is shown in Fig.2. This architecture combines the interaction of the users with the system which includes the frontend, backend, and the database. This System design combines a variety of technologies to give the Users (Admin, Supervisor, Student) a good user interface to interact with the Application.

Frontend: The responsive user interface of the application is developed using React framework, which provides seamless user engagement. Implementing the application in React framework improves the application's efficiency and enhances the user experience, in conjunction with React-bootstrap, Material UI and Modal. The visual appeal of the user interface of the application is further improved by using CSS styling.

Backend: The system's API framework is constructed using Express.js which simplifies the handling of server-side tasks and routing. Express.js smoothly integrates with Node.js, a reliable runtime environment for server-side operations. In order to facilitate smooth communication between the application and the MySQL database, the system incorporates Sequelize, a powerful library known for its capability to manage and manipulate data effectively. This combination of technologies ensures efficient data flow and interaction within the application.

This architecture design ensures a scalable, secure, and user-friendly experience for administrators, Supervisors and Students.

5.2 Use Case Diagram

This Use Case diagram depicts the visual representation of the application interaction between the actors and the System. In Fig.3 all the use cases of the application are shown, and a detailed explanation is given.

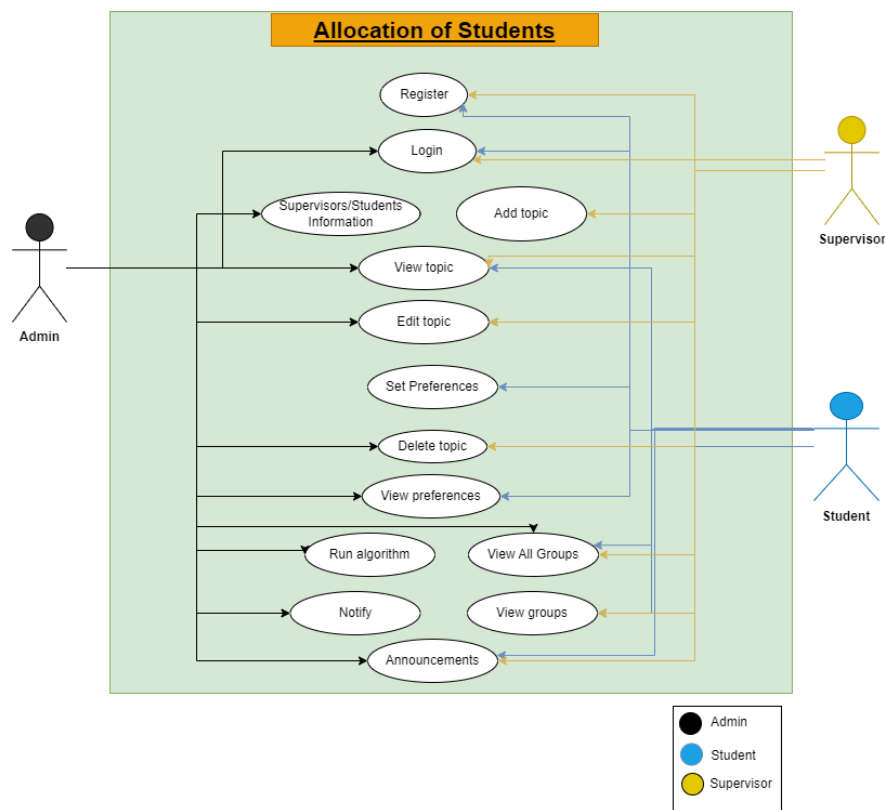


Figure 3: Use Case Diagram

The Actors of the application are Admin, Supervisors and Students. The high-level functionalities of the application developed are as follows based on the depicted diagram.

- **Register:** The users of the application Supervisor and Student create an account by registering their details such as Full Name, User email, Password, Department, Course and Role of the User.
Admin credentials are added to the database manually by the Author.
- **Login:** Users log in to the application by using the credentials that are given while registering their accounts. The system will authenticate the users by using JWT token authentication. The users can see their Dashboard once their login is successful.
- **View Supervisors/Students:** The administrator can view the list of all supervisors and students available with their information such as User email, Department and Course. The admin has the authority to edit/delete User details.
- **Add Topic:** The Supervisor is given permission to publish the topics that they are willing to supervise.
- **View Topic:** Topics that are published by the supervisor are made visible to the Admin, all Supervisors and Students.
- **Edit Topic:** The topics can be edited by the corresponding supervisor of the published topic or the Admin.
- **Delete Topic:** The published topics can be deleted by the specific supervisor or the Admin.
- **Set Preferences:** The Students are given permission to set of four preferences and the preferences can be changed until the deadline is over.
- **View Preferences:** The preferences set by the students can be viewed by the admin and the corresponding Student.
- **Run Algorithm:** Once the preferences are set by all Students and the deadline is over, the admin will execute the algorithm by setting the desired group size.
- **View All Groups:** Once the admin runs the algorithm by providing the group size, the algorithm will produce groups for all students, which is visible to Admin.
- **Notify:** Once the algorithm produces the groups, the admin can notify to students and supervisors about group allocation by sending them emails.
- **Announcements:** The admin makes announcements, whenever the announcements are published it is visible to students and supervisors. The admin can edit announcements and delete announcements.

5.3 Database Design and Schema

Database design is important in the development of the software application, as it involves structuring the database for efficient storage and access of data. The stored can be retrieved and manipulated as per the requirements of the application. A good database design helps in the integrity of the data, security of the application, and performance of the application as it increases the response time for the execution of a query.

Sequelize ORM is used to design database interaction using MySQL database. Sequelize helps in performing complex database queries, and the association of models. The database table is represented as models in Sequelize. The models and their relationships are defined with the help of association. Sequelize provides an efficient database operation performance.

The database of this allocation of students to Groups application consists of the tables shown in Fig.4. This database includes Users, Topics, Preferences, Groups and Group Users tables.

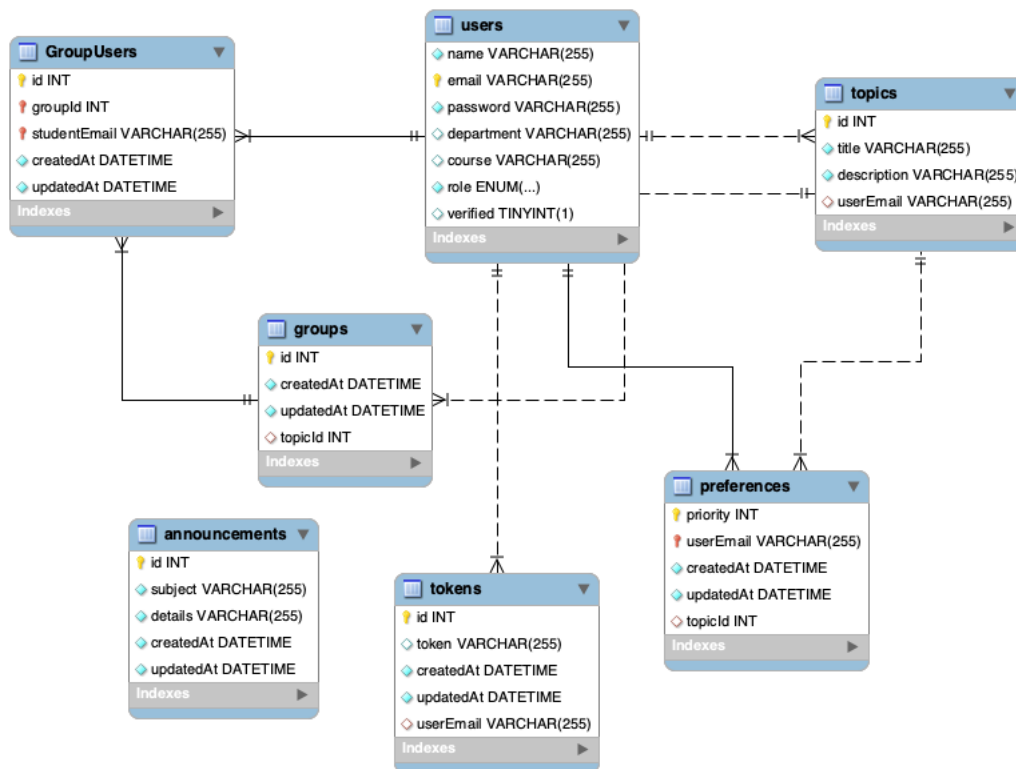


Figure 4: Database Design

Users: This table contains information about users which includes columns such as name, email, password, department, course, role and verified. The name column contains the full name of the users, the email column contains the valid email of the users which is the primary key, and the department column contains the various departments of supervisors and students. The course column contains the course of the students. The password column contains the password in the encrypted format where hashed passwords are stored for the security of the users. The role column contains the roles of the 3 different users which is admin, supervisors, and students. The verified column contains a Boolean value which is used to verify users of the application. It is set to false by default, when users are verified, the value is set to true.

Topics: The topic table contains columns which are id, title, description, userEmail. The id column contains unique id of the given topics. The title column includes the title of the topic, description column includes the description of the topic mentioned. The userEmail column contains the email of the supervisors of the corresponding topics. In this table userEmail is foreignkey referencing to Users table.

Preferences: This table contains columns such as priority, userEmail, createdAt, updatedAt, and topicId. The priority column contains the priority of the topics submitted by the students. Priority varies from 1 to 4. The userEmail column contains the email of the students who have submitted their preferences. createdAt and updatedAt column includes the timestamp of users, topicId column includes the unique id of the topics. In this table, userEmail and topicId are the foreignkey references to the tables Users and Topic respectively.

Groups: This table includes the groups of students. The table contains columns such as id, createdAt, updatedAt, and topicId. The id column indicates the unique identifier of each group, createdAt, updatedAt indicated the timestamps. The topicId column indicates the Topic Ids of the topics which is the foreignkey reference to the Topics table.

GroupUsers: This table contains the information of all students who are allocated to groups. It contains columns such as id, groupId, studentEmail, createdAt and updatedAt. Id refers unique id, groupId refers to the unique id given to each group, studentEmail contains the email information of each student who belongs to the group. createdAt and updatedAt creates the timestamp. GroupId and studentEmail are the foreignkey references to the tables Group and Users respectively.

Tokens: This table contains the information about token, it contains columns such as id, token, userEmail, createdAt and updatedAt. id refers to the unique identifier. The token column contains the token of the user and userEmail column contains the userEmail of the user which is the foreignkey reference to the Users table.

Announcement: This table contains columns such as id, subject, details, createdAt and updatedAt. The Id column contains a unique identifier for each subject. The subject column contains the name of the announcement subject, and the details column contains the details of the announcement.

5.4 Algorithm Design and Analysis

Fig.5 explains the flowchart of algorithm implementation. The time complexity of the algorithm is $O(N*M)$ where N is the number of students and M is the number of Topics.

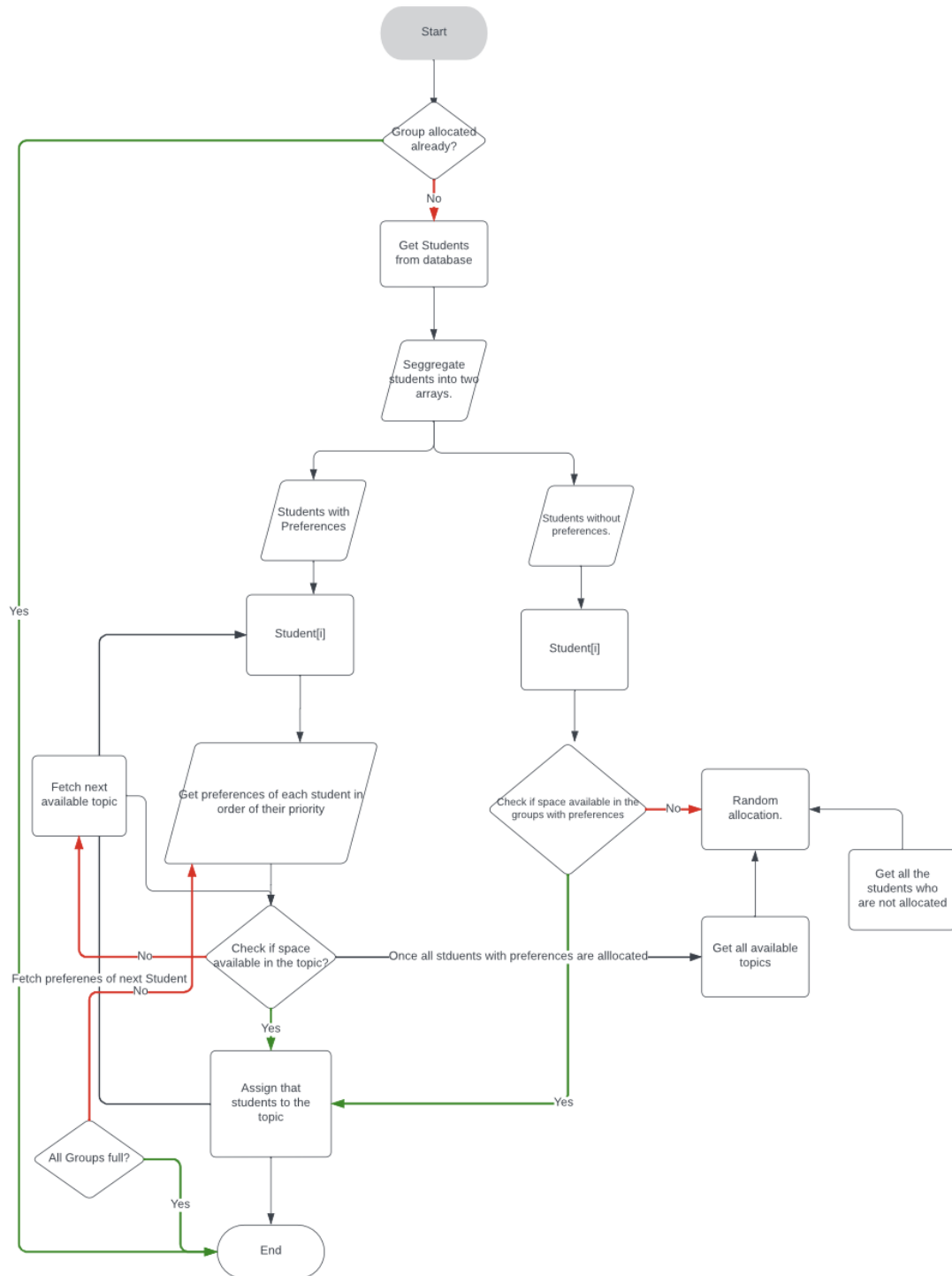


Figure 5: Algorithm flowchart

First, the maximum of the group is given, and the size of the group should be between 2 to 10, the allocation cannot be made. Then the algorithm checks if any groups formed already. If there is no allocation made, then in the first iteration of the algorithm, all the students from the database along with their preferences in order of priority are fetched. Then the students are ordered in the order of their timestamps which means the groups are allocated on a first come first serve basis. In the next iteration, the students are grouped into two, where one array consists of all the students with preferences and the other with all the students without preferences. First, the student with preferences is taken into consideration. To track the group availability and assignment of the topics two arrays are initialized. In the next iteration, the students with preferences are considered for allocation of the group. The loop is iterated for each student where it checks for the next available topic in the preferences given by the student. If a group for the topic exists, then the student is assigned to that group, and the group's availability is updated. The algorithm moves on to the next preferred topic if there is no group available for the current topic. The algorithm is then carried out for the students whose preferences are not given. These students are assigned to groups if there is any space available in the existing groups, or a new group is formed to allocate them to the topics randomly. The assigned groups are then stored in the database, if the admin is satisfied with the allocated groups, then the groups are notified to students and supervisors through email and as well the groups are made visible in the developed application. Considering all these iterations, the worst-case scenario for the given algorithm is determined to be $O(N*M)$ which is analyzed mainly from the Group Allocation iteration.

6 Project Outcome

6.1 Client-Side Modules

6.1.1 App Components

- **Welcome Page Component:** Welcome page is the first page of the application which serves as the entry point for users to access the application. The Welcome Page consists of registration and login navigation options for users to access the application.

- **Registration and Login Component:**

Registration: The registration page consists of a form to sign up for the application, which enables users to access the application features based on the permission given with respect to the user roles. Users are Supervisors and Students.

Fig.6 shows a registration page that has fields such as Full name, email address, department, and course. Here the user must specify if they are a student or Supervisor, once the user specifies the role, the user will be registered to the account accordingly. If the users are registered already, they can log in to the system with their credentials.

The image shows a registration form titled "Register". It contains the following fields and elements:

- Full Name:** A text input field with the placeholder "Enter your Full Name".
- Email Address:** A text input field with the placeholder "Enter your Email Address".
- Department:** A text input field with the placeholder "Enter your Department".
- Course:** A text input field with the placeholder "Enter your Course".
- Role Selection:** Two radio buttons labeled "Supervisor" and "Student".
- Register Button:** A blue button with the text "Register".
- Login Link:** A link labeled "Registered? Login" located at the bottom right of the form.

Figure 6: Registration Page

Supervisor: The Supervisor must register to the account by selecting the supervisor option, and then entering the details such as Full name, Email Address, Password, and Department. Once the user registers to the account, they will receive an email to verify which consists of a link for verification, once the user clicks on the link and verifies themselves, they will be registered to the account, and they can access the application by logging in to the application by entering their E-mail address and Password.

Student: Students will register to the account by selecting the student option and entering the fields Full Name, Email Address, Password, Department and Course. Once the Student verifies themselves, they can access the application by logging in to the account with their credentials.

The email verification is done for the security purpose of the application.

Login: Once the Users are registered to the account, they can log in to the application by navigating to the Login Page, which consists of a form which has the user Email Address and Password fields, when the user enters their email address and password, a token will be generated, this token is used to verify the user details that are stored in the database, if the token is valid, the user will be redirected to their respective dashboard. If the token isn't verified, then the user is unauthorized to access the application.

User Dashboard:

Once the user is logged in, they will be redirected to the dashboard which consists of different features provided in the navbar which helps in navigation. The navbar has user-role- specific features.

Admin Dashboard:

Admin dashboard consists of the following functionalities:

- **Home:** The Home component consists of an admin Homepage which has an Announcement component. The announcement component consists of two components MakeAnnouncement and AllAnnouncements.

Make Announcements component, which will allow the admin to notify events to supervisors and students on their respective homepage. This will have subject and details fields, when the admin uploads the announcement, it is displayed on the Supervisor and Student's Homepage.

All Announcements component, Announcements made by admin will be displayed under this, the latest announcement being on the top with the date included. The admin can also edit the announcements made if there are any changes required.

- **View Topics:** This component consists of a list of all the available topics published by the supervisor. This consists of a table with the topic code, the title of the topic, the description of the topic and the name of the corresponding supervisor. This also consists of a search bar which will help in filtering topics that provide an easy way to find topics.
- **Update Topics:** This component allows the admin to edit the topics uploaded by the supervisor. This has an edit option which upon clicking displays Modal to edit the topic details and save the changes. The admin can edit all the topics uploaded by the supervisors.
- **Delete Topic:** This component displays all the topics with the delete button which upon clicking will ask for confirmation to delete, if the admin confirms to delete the topic, the topic will be deleted from the database.
- **View Students:** This component allows the admin to see the list of all available students with their information about the course, department, email address, edit option and student preferences. The admin is enabled with the permission to edit student details or to delete any students from the records. A search bar is provided to filter students by their names, this feature allows admin to find student records easily. The Student Preference column provides information about a particular student's preferences. This consists of a Modal that contains students' current preferences and an option to edit student preferences when any student has any issue submitting their preferences or updating them. Admin has the option to set deadlines. The admin can set the deadline which disables the submission of topic preferences for all the students.

Fig.7 shows the preferences of students in the admin dashboard, where the admin can see the preferences set by students. The admin can also add or update student preferences on behalf of students by clicking the 'Edit' button.

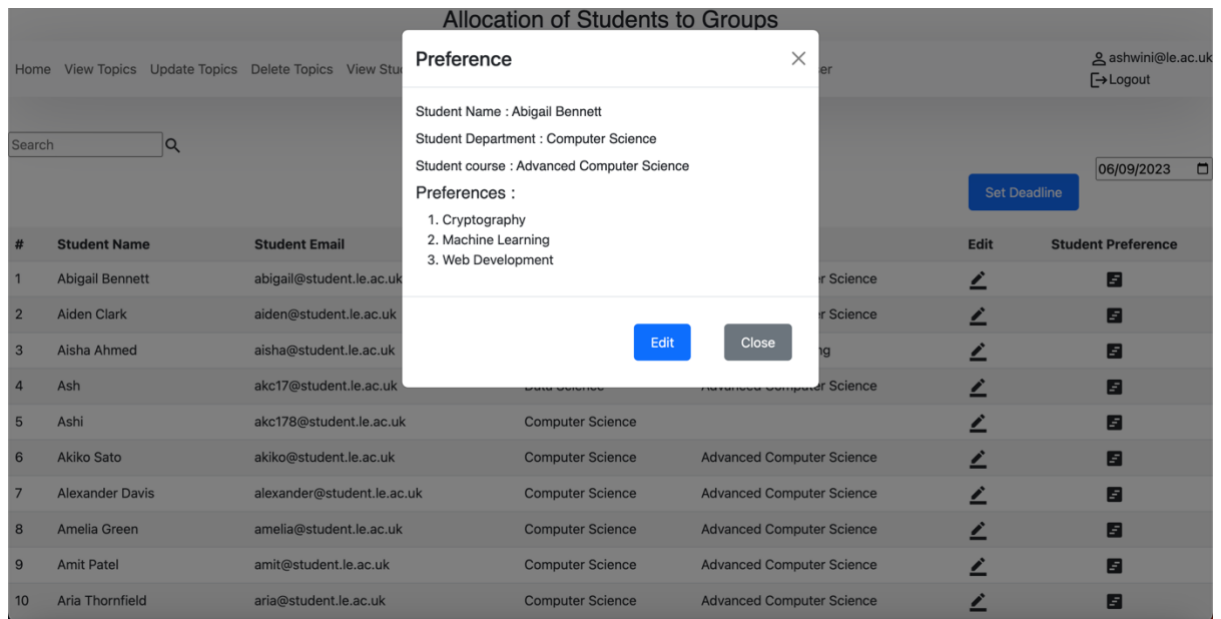


Figure 7: Student preferences on admin page

- **View Supervisors:** This component displays the records of all the supervisors. This consists of information in a table that includes the Supervisor's name, Email address, and Department. This consists of a search bar which allows the admin to filter supervisors' records by their name. The admin can edit or delete supervisor records.
- **Allocate Students:** This component allows the admin to set the group size for students. The admin can change the size of the groups according to the requirement. Upon admin allocating the group size, this will run the matching algorithm and form a group consisting of the topic name and the corresponding topic supervisor's name. When the admin allocates groups the Students and the Supervisors will be notified with an email about topic allocation which includes the group number, topic name, name of all the students in the group and the corresponding topic's supervisor name.
- **View Groups:** This component consists of information on all the groups that are produced by running the algorithm. This includes the group number for each group, the title of the topic assigned, the name of all students and their supervisor's name. The admin can view all the groups produced by the algorithm, if the admin is not satisfied with the results or the group size, all groups can be deleted and the groups can be allocated again by providing the group size. The group size can't be given 0 or 1, if the size of the group is less than 2 then an error will be thrown. If the topics are insufficient to allocate all the students, then an error will be thrown saying the topics are insufficient. Once the groups are allocated, the admin will see the result, if the admin is satisfied with the results, all students and supervisors will be notified accordingly.

Supervisor Dashboard:

- **Home:** This component consists of a supervisor homepage, which includes an Announcement component under the notification heading, where the announcements made by the admin are displayed. The latest announcements are displayed at the top of the list.
- **Add Topic component:** This component allows the supervisor to upload topics, it consists of a card which has a title and description, the topics can only be published by the supervisors. Once the topics are published it is stored in the database and the successfully topic uploaded message will be displayed. The supervisor can't publish an empty topic title or description; it will throw an error.
- **View Topic Component:** This component enables the supervisor to view all the topics that are published by all supervisors. This consists of a table that includes the topic code, title, description of the topic and the corresponding supervisor email address. This included a search bar, which allows the supervisor to filter the topic by its name.
- **Update Topic Component:** This component allows the supervisor to edit the topic that is published by them, the supervisor can only edit the topic that is uploaded by them and is unauthorized to edit topics that aren't published by them. A modal is displayed when the update button is clicked, the modal consists of the topic name and its details to make any changes whenever required. The Fig 8. Displays Supervisor Update Topic page where the supervisor is displayed with topic name and topic description which consists of uploaded topic details that can be edited as required.

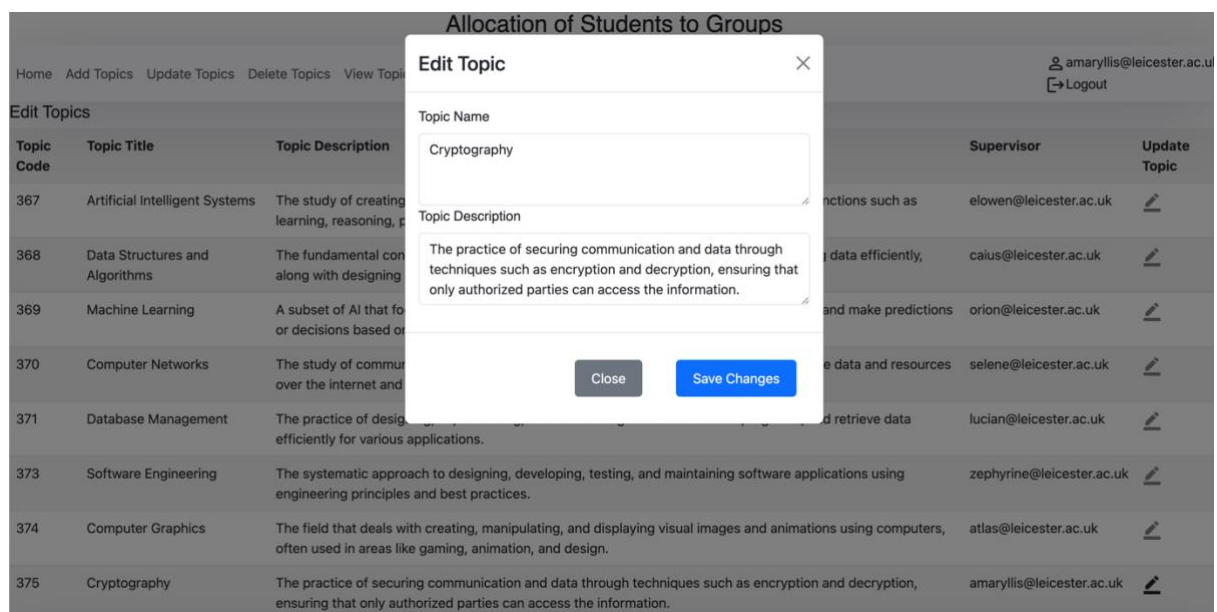


Figure 8: Supervisor Update Topic page.

- **Delete Topic Component:** This component allows the supervisor to delete a topic that is been uploaded. The supervisor is given permission to delete only those topics that is published by them. A delete button is provided which upon clicking will ask for confirmation to delete the topic, if the supervisor confirms, that topic will be deleted from the database.
- **View Groups Component:** This component allows the supervisor to view the groups that have been assigned to them. This group consist of information such as group number, the title of the topic, the email address of all the students who are assigned to that group and the supervisor's name.

Student Dashboard:

- **Home Component:** This component consists of the Home page which has an Announcement component that displays the announcements that are made by the admin, the latest announcement is displayed at the top.
- **View Topic Component:** This component consists of a list of all the topics that are published by supervisors. The topic information is displayed in a tabular column which consists of the topic code, topic title, topic description, and email address of the corresponding supervisor. This component is provided with the search bar which allows the students to search through the topics by the topic names easily. If the student tries to search for any topics that aren't published, the search result is displayed with No records found message.
- **View Preferences Component:** This component allows the student to set preferences, the preference table consists of the topic code, topic name and the supervisor's email address. The student should set 4 preferences, upon submitting the preferences, it is displayed under the heading, current preferences. The students can change preferences any number of times until the deadline is over. Fig.9 shows the preferences page where a student can set preferences in order of priority. The preferences that are set by a student will be displayed under Current Preferences which helps student to track preferences and change the preferences if required.

[Home](#)
[View Topics](#)
[Preferences](#)

akc17@student.le.ac.uk
 Logout

Preferences

Current Preference

1. Cryptography
2. Software Engineering
3. Computer Networks
4. Distributed Systems

Topic Code	Topic Title	Supervisor	Preferences
367	Artificial Intelligent Systems	elowen@leicester.ac.uk	Select your option
368	Data Structures and Algorithms	caius@leicester.ac.uk	Select your option
369	Machine Learning	orion@leicester.ac.uk	Select your option 1 2 3 4
370	Computer Networks	selene@leicester.ac.uk	Select your option
371	Database Management	lucian@leicester.ac.uk	Select your option
373	Software Engineering	zephyrine@leicester.ac.uk	Select your option
374	Computer Graphics	atlas@leicester.ac.uk	Select your option
375	Cryptography	amaryllis@leicester.ac.uk	Select your option
376	Web Development	amaryllis@leicester.ac.uk	Select your option
385	Distributed Systems	thalia@leicester.ac.uk	Select your option
391	Opinion Mining in Social Media2112123	amaryllis@leicester.ac.uk	Select your option

Submit

Figure 9: Student Preferences page

- View Group Component:** This component consists of group information of allocation, once the admin runs the matching algorithm by providing the group size, the allocated group information will be displayed in this component. The group information is displayed inside a card, the student can see the group number, the group topic, the email address of the corresponding topic's supervisor and the names of all other students who are in the same group.

6.2 Server-Side Modules

The backend application is developed using Express.js which is used to handle the requests and responses from HTTP, routing, and middleware components. The Controller component is used to define the server-side logic of the application. Middleware functions are written to control the user's access to the application and for the authentication of the application. For example, the 'adminVerify' middleware is written to verify if the user who is authenticated to access application has the role 'Admin' to access the functionalities of the admin. This middleware helps in providing specific routes for user access to access specific functionalities of the application. This has made the application easy to provide user access to admin, supervisor, and student. The Models are written to define the database relation between the tables. Models are written using Sequelize ORM in relation to MySQL database. Routes are specified in a separate component that helps maintain and access the routes easily, utils component is created to manage the general functions of the application such as email service and password encryption functionalities. All the modules that are written in the backend in shown in Fig.10 below.

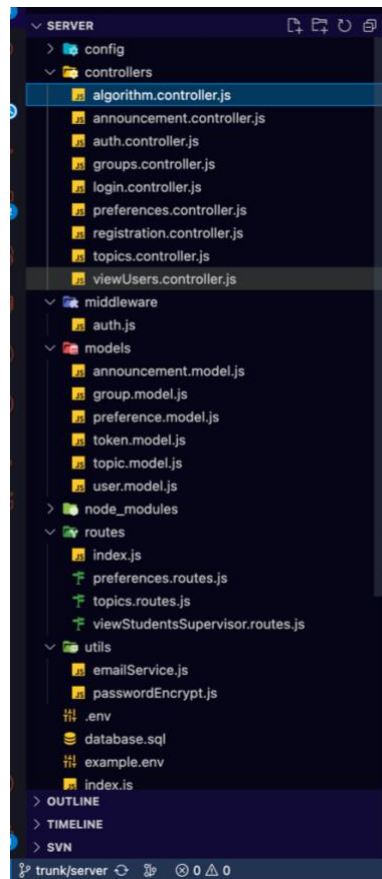


Figure 10: Server-Side Modules

In the backend, REST API is used to communicate with the server, this promotes stateless communication between the client and server over HTTP protocol. In this CRUD (Create, Read, Update, Delete) operations are used which include several methods such as the GET method to retrieve information, the POST method to create new information, the PUT method to update existing resources and the DELETE method to delete resources.

The below Fig.11 is an expected example response of a GET request to this URL “api/preferences” with a query parameter userEmail. If the query parameter userEmail is not passed with a specific student email, then all the student preferences are listed which is shown in Fig. 12

```
[
  {
    "priority": 1,
    "topic": {
      "title": "Artificial Intelligent Systems",
      "id": 367,
      "description": "The study of creating intelligent agents or systems that can mimic human-like cognitive functions such as learning, reasoning, problem-solving, and decision-making."
    }
  },
  {
    "priority": 2,
    "topic": {
      "title": "Data Structures and Algorithms",
      "id": 368,
      "description": "The fundamental concepts in computer science that deal with organizing and manipulating data efficiently, along with designing algorithms for solving computational problems."
    }
  },
  {
    "priority": 3,
    "topic": {
      "title": "Machine Learning",
      "id": 369,
      "description": "A subset of AI that focuses on developing algorithms that enable computers to learn from and make predictions or decisions based on data, without being explicitly programmed."
    }
  },
  {
    "priority": 4,
    "topic": {
      "title": "Computer Networks",
      "id": 370,
      "description": "The study of communication systems and protocols that enable digital devices to exchange data and resources over the internet and other interconnected networks."
    }
  }
]
```

Figure 11:JSON response for GET Request

Example showing Preferences of few Students in JSON format,

```
{
  "priority": 1,
  "topic": {
    "title": "Artificial Intelligent Systems",
    "id": 367,
    "description": "The study of creating intelligent agents or systems that can mimic human-like cognitive functions such as learning, reasoning, problem-solving, and decision-making."
  }
},
{
  "priority": 1,
  "topic": {
    "title": "Artificial Intelligent Systems",
    "id": 367,
    "description": "The study of creating intelligent agents or systems that can mimic human-like cognitive functions such as learning, reasoning, problem-solving, and decision-making."
  }
},
{
  "priority": 1,
  "topic": {
    "title": "Artificial Intelligent Systems",
    "id": 367,
    "description": "The study of creating intelligent agents or systems that can mimic human-like cognitive functions such as learning, reasoning, problem-solving, and decision-making."
  }
},
.....
```

Figure 12:JSON response for GET Request

A list of all the backend APIs used by the Admin, Supervisors and Students are listed in Table 3.

Table 3: API Routes for all functionalities implemented.

APIs	Method	Controller	URL Endpoint	Request body parameters in JSON
User Registration	POST	API used to register user accounts.	api/register	name,email,password,department,role, verified.
User Verify	POST	API used to verify users.	api/token/verify	token, userEmail
User Login	POST	API used to login to account.	api/login	email, password
Token Verification	GET	API used to verify the token.	api/verify/token	token.
Topics.	POST	API used to post topics, to get topics, to get topics by topic id, to delete topics and to edit topics. To search topics by title.	api/topics	token, title, description.
	GET		api/topics	token
	GET		api/topics/id	token
	DELETE		api/topic/id	token
	PUT		api/topics/id	token, title, description.
	GET		api/topics/searchTopics?title={Topic title}	token
Preferences	POST	API user to submit all preferences, to get preferences of specific student and to get preferences of all students.	api/preferences	token, first preference, second preference, third preference, fourth preference.
	GET		api/preferences?userEmail={student email}	token
	GET		api/preferences	token
Student data	GET	API used to view all student information, to filter students by their name, to delete students and to edit student information.	api/viewStudents	role
	GET		api/user/searchStudents?name={Student name}	token
	DELETE		api/deleteStudents	token, email
	PUT		api/user	token,name,email,course,department
Supervisor data	GET	APIs used to view all supervisor information, to filter supervisors by the name, to delete supervisors and to edit supervisor information.	api/viewSupervisos	role
	GET		api/user/searchSupervisors?name={Supervisor name}	token
	DELETE		api/deleteSupervisors	token,email
	PUT		api/user	token,name,email,department
Groups	POST	APIs used to allocate size of group, to get all	api/allocate	token,groupsize
	GET		api/groups	NA

	GET	groups allocated, to get group specific to a student, to get groups specific to a supervisor, and to delete the allocated group	api/groups/student	token
	GET		api/group/supervisor	token
	DELETE		api/groups	token
Announcements	POST	APIs used to make announcement, to display announcements made, to edit announcement and to delete announcements.	api/announcement	token,subject,details
	GET		api/announcement	token
	PUT		api/announcement/id	token,subject,details
	DELETE		api/announcement/id	token

6.3 Code Snippets

```
const findTopicsByPriority = async (req, res) => {
  try {
    const maxGroupSize = Number(req.body.groupSize) || 5;
    // console.log(maxGroupSize);
    if(maxGroupSize < 2 || maxGroupSize > 10 ){
      return res.status(400).json({error: "Group size should be more than 2 and less than 10"})
    }
    const groupsExist = await Group.findAll();
    if(groupsExist.length > 0){
      return res.json({ message: "Groups already formed."})
    }

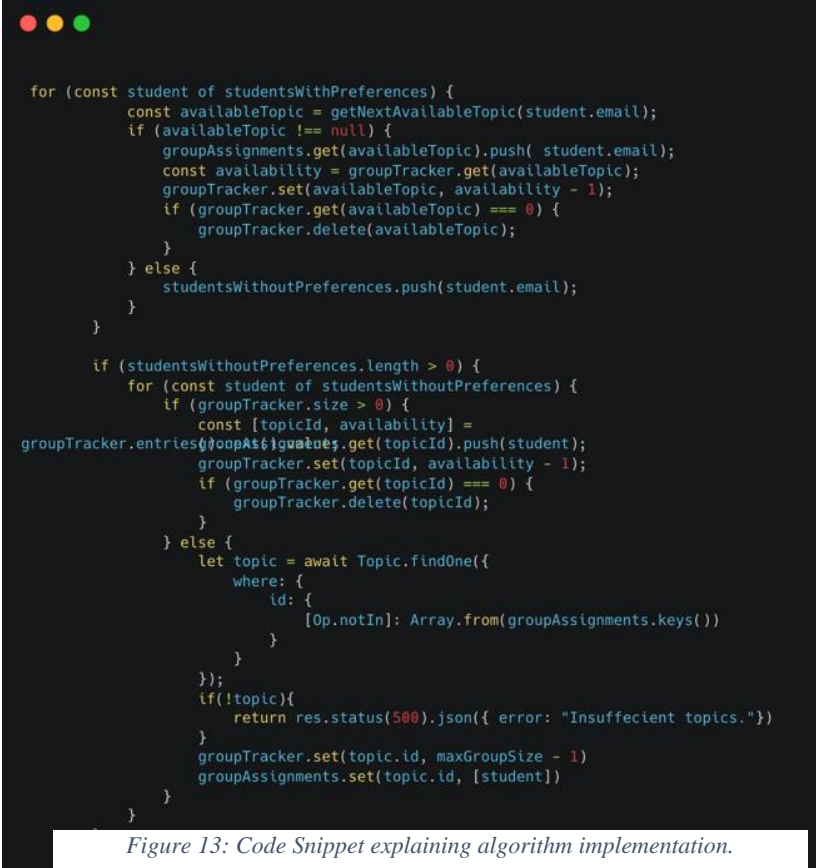
    const students = await User.findAll({
      where: { role: 'Student' },
      include: { model: Preference, as: 'preferences', order: [['priority', 'ASC']] },
      order: [[{ model: Preference }, 'updatedAt', 'ASC']]
    });

    const studentsWithPreferences = students.filter(student => student.preferences.length > 0);
    let studentsWithoutPreferences = students.filter(student => student.preferences.length === 0);
    studentsWithoutPreferences = studentsWithoutPreferences.map(student => student.email);

    const studentPreferences = new Map();
    studentsWithPreferences.forEach(student => {
      let preferences = student.preferences.sort((a,b) => a.priority-b.priority)
      studentPreferences.set(student.email, preferences);
    });

    const groupAssignments = new Map();
    const groupTracker = new Map();

    const getNextAvailableTopic = (studentEmail) => {
      const preferences = studentPreferences.get(studentEmail);
      for (const preference of preferences) {
        if (!groupAssignments.get(preference.topicId)) {
          groupAssignments.set(preference.topicId, []);
          groupTracker.set(preference.topicId, maxGroupSize);
        }
        if (groupAssignments.get(preference.topicId).length < maxGroupSize) {
          return preference.topicId;
        } else {
          groupTracker.delete(preference.topicId);
        }
      }
      return null;
    };
  }
};
```

```

for (const student of studentsWithPreferences) {
  const availableTopic = getNextAvailableTopic(student.email);
  if (availableTopic !== null) {
    groupAssignments.get(availableTopic).push( student.email);
    const availability = groupTracker.get(availableTopic);
    groupTracker.set(availableTopic, availability - 1);
    if (groupTracker.get(availableTopic) === 0) {
      groupTracker.delete(availableTopic);
    }
  } else {
    studentsWithoutPreferences.push(student.email);
  }
}

if (studentsWithoutPreferences.length > 0) {
  for (const student of studentsWithoutPreferences) {
    if (groupTracker.size > 0) {
      const [topicId, availability] =
groupTracker.entries().next().value;
      groupTracker.set(topicId, availability - 1);
      if (groupTracker.get(topicId) === 0) {
        groupTracker.delete(topicId);
      }
    } else {
      let topic = await Topic.findOne({
        where: {
          id: {
            [Op.notIn]: Array.from(groupAssignments.keys())
          }
        }
      });
      if (!topic){
        return res.status(500).json({ error: "Insufficient topics."})
      }
      groupTracker.set(topic.id, maxGroupSize - 1)
      groupAssignments.set(topic.id, [student])
    }
  }
}

```

Figure 13: Code Snippet explaining algorithm implementation.

Code Snippet 1 Fig.13 shows the implementation of the matching algorithm. This algorithm is implemented considering the stable marriage algorithm, this is the main component of this application, which is designed to allocate students to groups based on their topic preferences. The initial part of the code initializes the maximum size of the group. If the size of the group is not specified between the range 2 to 10 then the code will return a 400 Bad Request response. Next, it checks for the existing groups, if the groups are already formed it responds with a message saying that the groups have already been formed. The code then retrieves student information from the database, focusing primarily on those with the role “Student”. This also retrieves the associated preferences of the students from the database by making certain that the preferences are retrieved according to the order of student priority. To facilitate the allocation process, it filters the students into two arrays ‘studentWithPreferences’ and ‘studentWithoutPreferences’. Students with preferences are identified by filtering the dataset based on the preference count. The email addresses of the students without preferences are stored, these students will be allocated to a different group from those who have set preferences. It organizes the preferences of students who have preferences into a map, where the key is the student's email and the value is a list of those preferences, ordered by priority within an array. Next, it initializes the data structure to two maps, ‘groupAssignments’ which keeps track of students who are assigned to which topics and ‘groupTracker’ which keeps track of the availability of group space for each topic. The main allocation logic is written in ‘getNextAavailableTopic’. This method iterates over a student's preferences and looks for vacant spots within topic groups after receiving the student's email. This method returns the

corresponding topic id, if a place is determined to be available, else it returns null. Then the subsequent section assigns students to topics. Subsequently, the code iterates through students with preferences and assigns students to topics based on their preferences this also ensures that the group size is not exceeding the maximum group size for each size. Then, it checks for students without preferences and checks if there are any vacant spaces available, if there are spots available then it assigns the students without preferences to those topic groups. In the end, the code creates a group for each topic by assigning students to that group with the corresponding topic supervisor.

```

router.post("/verify/user", async (req, res) => {
  const token = req.body.token;
  const userEmail = req.body.userEmail;

  const dbToken = await Token.findOne({
    where: {
      userEmail
    }
  });

  if(! dbToken){
    return res.status(500).json({ error: "Something went wrong. Please register again."})
  }

  const isMatch = dbToken.token == token;

  if (isMatch){
    await User.update({ verified: true }, {
      where: {
        email: userEmail
      }
    })

    res.json({
      message: "User verified.",
      verified: isMatch
    })
  }
  else{
    await User.destroy({
      where: {
        email: userEmail
      }
    });

    res.json({
      message: "User not verified. Please register again.",
      verified: isMatch
    })
  }
})

```

Figure 14: Code Snippet explaining Registration and email verification.

This code snippet 2 in Fig.14 explains the user registration part of the web application. This code extracts user registration information from the request body which includes the user's name, email, password, course, department, role, and verified status. Then it checks whether the email format is valid based on the user's role. Students should have email addresses ending with "@student.le.ac.uk", supervisors should have addresses ending with "@leicester.ac.uk". If the email format is incorrect then an error message is displayed which returns 400 Bad Request responses. The user password is hashed for storing passwords securely in the database using the function "hashedPassword()". The verified filed is initially set to false, which indicates the user is not verified, it is set to true when the user verifies. Random token will be created using 'crypto' module. This generated token is used for email verification. The generated token is stored in the database along with the user's email, which is used for the verification of user. Email verification message is constructed which includes a verification link to verify user's email address. The email is sent to the users using the "emailSender" function which uses email service. When the users verify the link sent to the email, registration

is successful which returns user's information which includes name, email, department, course and role. This code contains error-handling messages for various scenarios, such as validation errors, unique constraint violation error, and database errors.

The second part of the snippet explains the route for email verification, where the token and user email are extracted from the request body. This queries the database to check for the matching token that is associated with the user email. If the token is found, then it verifies whether the token provided matches the token stored in the database. Otherwise, it requests the user to re-register. When the token is verified, this updates the User's verified status as true. If the user verification is unsuccessful then it cancels the user registration. Incorporating the user email verification provides security allowing only those with verification to access the account.

```
const postPreference = async (req,res)=>{
  let user=req.user
  if(user.role == 'Admin'){
    user.email = req.body.studentEmail
  }
  const existingPreferences = await Preference.findAll({
    where: {
      userEmail: user.email,
    },
  });

  const { first,second,third,fourth } = req.body

  if(!first || !second || !third || !fourth){
    return res.status(400).json({error:"Plese select 4 preferences"})
  }
  const priorityOneTopic=await Topic.findByIdPk(first)
  const priorityTwoTopic=await Topic.findByIdPk(second)
  const priorityThreeTopic=await Topic.findByIdPk(third)
  const priorityFourTopic=await Topic.findByIdPk(fourth)

  if(!priorityOneTopic || !priorityTwoTopic || !priorityThreeTopic || !priorityFourTopic)
  return res.status(400).json({error: "Invalid topics."})

  if (existingPreferences.length > 0) {
    await Preference.update({priority: 1, userEmail: user.email, topicId: first }, {
      where: {
        userEmail: user.email,
        priority: 1
      },
    })
    await Preference.update({priority: 2, userEmail: user.email, topicId: second }, {
      where: {
        userEmail: user.email,
        priority: 2
      },
    })
    await Preference.update({priority: 3, userEmail: user.email, topicId: third }, {
      where: {
        userEmail: user.email,
        priority: 3
      },
    })
    await Preference.update({priority: 4, userEmail: user.email, topicId: fourth }, {
      where: {
        userEmail: user.email,
        priority: 4
      },
    })
    return res.json({ message: "Updated Succesfully"});
  }

  const preferences = await Preference.bulkCreate([
    {
      priority: 1,
      userEmail: user.email,
      topicId: first
    },
    {
      priority: 2,
      userEmail: user.email,
      topicId: second
    },
    {
      priority: 3,
      userEmail: user.email,
      topicId: third
    },
    {
      priority: 4,
      userEmail: user.email,
      topicId: fourth
    }
  ]);

  const studentPreferences = await User.findByIdPk(user.email, { include: [
    {
      model: Preference,
      attributes: ['priority'],
      include: { model: Topic, attributes: ['title', 'id', 'description'] },
    },
  ], })

  res.json({ message: "Preferences set."})
}
```

Figure 15:Code Snippet explaining Preferences.

The code snippet 3 in Fig.15 explains the process of setting the preferences for topics by students. The initial part of the code authenticates the user, if the user is an Admin, it allows the admin to specify preferences on behalf of students. Next, it checks if the student already has existing preferences that are stored in the database, if the student preferences already exist in the database, then the code updates the student preferences based on the topic priorities provided by the student in the request. If the preferences are not found in the database for that student, then it creates a new preference record for the student. Next, the code makes sure all four preferences are provided in the request and that the given topic preference exists in the database. If any given condition is not satisfied, then the code throws an error message returning a Bad Request response. After setting the preferences or updating the preferences, the code responds with a JSON message that the preferences are successfully created or updated.

6.4 Libraries and Plugins

Table 4 below, the list of all the Libraries and Plugins that are used in the development of this web application is written. The built-in libraries provide resources that simplify the development process and help in building the application more efficiently.

Table 4: Libraries and Plugins

Library Name	Usage
Material UI	Used in the front-end application framework to design icons to navigate to different functionalities of the application.
Bootstrap	Used in the front-end to design the components of the application and to design the tables.
MySQL	Used in the backend to design database to interact with the server.
Sequelize	Used in the backend as Object Relational Mapping tool, which is used to interact with the relational database which is MySQL.
Dotenv	Used in the backend to store database details, email connection string and secret keys.
JSON Web Token	Used in the backend to generate token for users.
Azure Communication Email	Used in the backend for email communication.
Bcrypt	Used in the backend to hash passwords.
Body Parser	Used in the backend to parse HTTP request bodies.
Nodemon	Used in the backend to detect changes in the file and restart the server automatically.
Axios	Used in the front-end to handle responses from server.
React-router-dom	Used in the front-end to handle navigation of components and to handle routing.
CSS	Used in the front-end to style application.

7 Software Testing

Software testing is carried out to test the application development. Testing is done to detect issues or errors in the application, this allows fixing bugs in the application before deploying the application. In the development of this application, testing is done parallelly.

Functional Testing: To verify the core functionalities of the application such as login, preference selection and submission, email notifications, and visibility of allocated topics, supervisors, and group this test is carried out. Verification of the algorithm to check if the algorithm is working correctly in assigning students to groups, topics, and supervisors this testing is carried out. The functionality of each component is tested by giving positive and negative scenarios. For example, When users register to an account, an email should be sent to verify the users, if the email is not verified then the users can't register to the account or login to the system. When users register to the account, the password length must be 8 digits, else the system won't allow users to register to the account.[11]

Integration Testing: This test is carried out to check for the integration of components to verify if all functions are working as expected. When components are developed, the flow between the two components is tested. For example, when the Admin allocates students to a group, the allocation is made visible to Supervisors and Students, one more scenario is when the admin makes any announcement, it is made visible to Supervisors and Students.

User Interface Testing: This test is carried out to verify if the application is user-friendly and responsive. The application is thoroughly tested to verify Buttons, Forms and navigation between the pages of the application. The application is also tested to check for the responsiveness of the Modal.

API Testing: The application is tested using the Postman tool for API testing, this tool is used to perform thorough testing on the application's functionality to receive the expected responses. All the user routes have been created and tested to check if the expected result is produced. During the API testing course, many issues were raised that required debugging and correction in the code. This testing played a major role in identifying bugs in the application which helped the author to fix issues to enhance the development of the application.[12]

The below Fig.16 shows the test case and response of a case where all the Student's information is shown, in the request body, the role of the user is passed which fetches the details of the user. In the Fig.15 the role is passed as 'Student', this retrieves information about all students. The postman takes the token which is the admin's token, if the token is invalid, it throws an error message as 'Unauthorized'. Similarly, all the user routes for all the functionalities are tested using this tool to check for expected results.

Fig.17 shows the scenario where when an empty topic description is passed in the request, an error message will be returned which says "Fields cannot be empty".

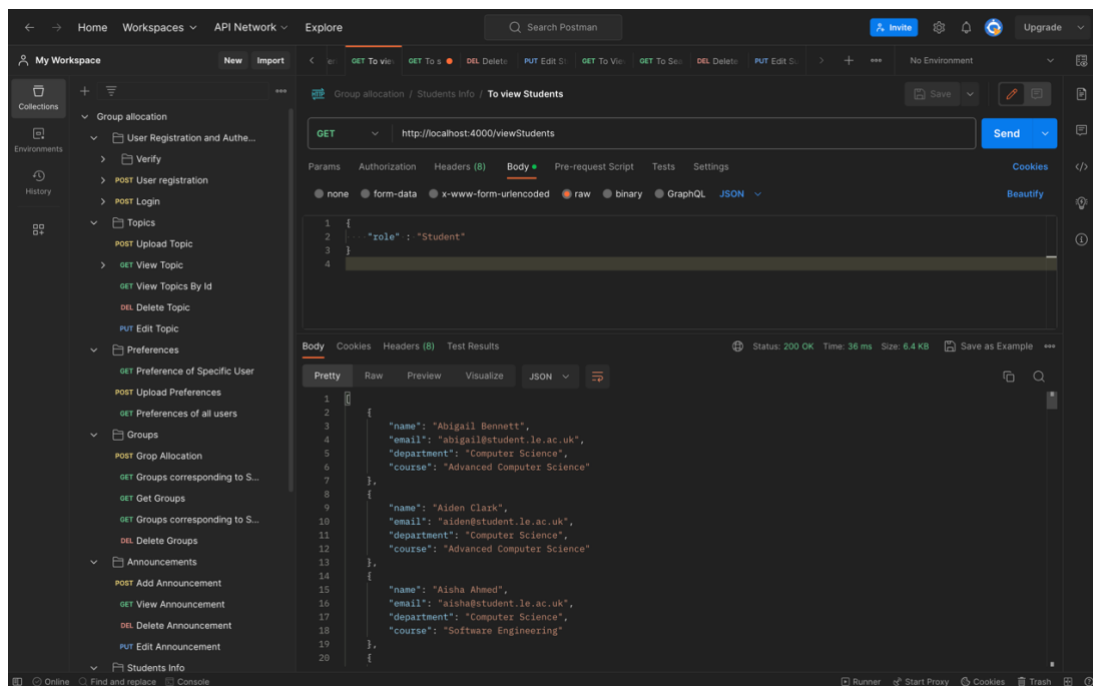


Figure 16: API testing in Postman displaying all students.

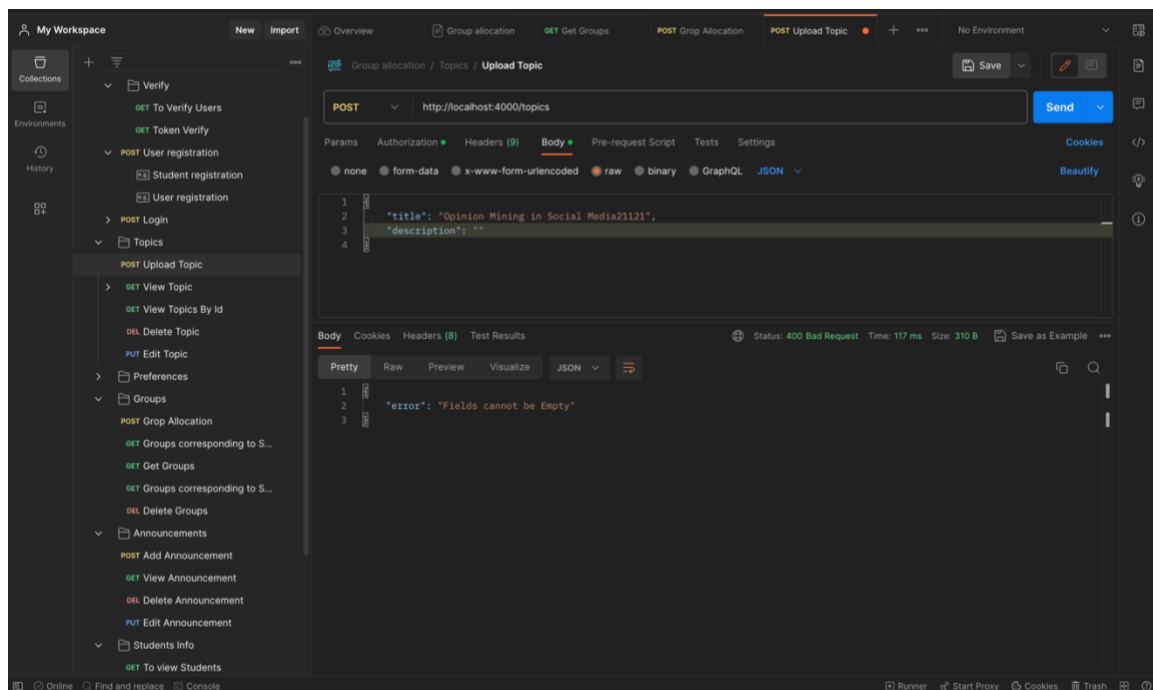


Figure 17: API test showing negative case scenario.

8 Conclusion

8.1 Challenges Faced

The development of the application was started as per the plan, Since the author is new to the React framework, the development of the frontend features consumed more time than the time plan given.

The ORM used to relate the MySQL, which is Sequelize is new to the author and was difficult to learn and implement in the given timespan. The documentation on Sequelize is a little confusing and many other resources had to be referred to understand the associations between the tables and querying the database.

A few of the application features were modified and new features were added to the time plan after the feedback received from the interim interview with the Supervisor. This modification changed the time plan as the application had to be modified accordingly.

- User registration by admin. The author had implemented a Registration page for self-registration by Students and Supervisors. After the Interim interview feedback, the User registration by admin is also included.
- Filter topics, students and supervisors.
- Password Strength check has been included.
- Email verification has been done.
- Group size change by admin is implemented.

All these features mentioned in the feedback have been implemented by the author.

The implementation of the algorithm to allocate students to groups required more time as this was the core feature of the application, decide on a suitable algorithm as per the requirement took more time, as it required many research papers to be referred to understand the allocation process and to decide on which matching algorithm best suits the given requirement. The algorithm required many modifications and fixes. Researching many matching algorithms consumed more time.

Implementation of the email verification service consumed more time as the author had decided to implement using AWS service at the beginning, due to the issues that arose later in setting up the service, the author decided to change the email service to Azure communication service, this consumed more time to change from one email service to another and this caused a delay in the time-plan. While building the email service, the author faced many issues in understanding and implementing the service.

During the testing of the application, many issues were discovered and needed fixing. The application has been tested carefully considering all scenarios to provide a working application.

8.2 Feedback

The application developed has been improved by gathering feedback from the author's friends at the university. The application was explained to them with all the functionalities that had been implemented, and they explored the application and provided feedback. They also helped by creating their account and registering themselves, once they explored the application, they provided feedback which helped in improving the application. While taking the feedback, a few errors in the application were noticed that required proper error display messages, which have been fixed by displaying the proper error message responses.

A few feedback points that were given are mentioned below:

- The upload topic was letting supervisors submit empty topics, this was missed by the author while testing the application, this was noticed while gathering feedback and the issue was fixed. This test case helped in solving accidental entries to topic records.
- Email service verification was tested by adding the author's friends account, and verifying if proper group allocation mails are sent after the admin notified to send allocation details to Students. The allocation was sending emails right after the allocation without the admin's permission this was resolved by adding a notify students button.
- The registration form was letting students submit the form without specifying the course, this was resolved by making that field as required as it is necessary to verify that a student is eligible to register for a specific course.

The feedback has helped the author to get different perspectives on application development.

8.3 Future Improvement

The application has been designed with all the essential and recommended requirements that are provided. In the future, the application could be extended to have the deadlines integrated into the Calendar system. This helps students to keep track of the event deadlines when integrated with the Calendar.

The application could be extended to have a discussion forum for students to discuss the module, once the students receive group information. This would help students to communicate with each other and discuss their ideas related to topics assigned to them. This would also help students to create threads for various parts of the module that help in organizing the topic information and accessing the contributions that are made by other members of the group. This would also help students to view information from previous discussions, which will help students keep track of all the valuable information that is shared among them about the module. The students could discuss their issues related to the module which could have solutions provided by other students of the group. The author had time constraints to implement this feature as the knowledge to build a discussion forum was limited and the application completion time frame was short.

The application could also be extended to include an Accept/Reject topics allocation feature in future improvement. This would help students who are unsatisfied with the group assigned to them to reject the allocation. This feature requires the admin to remove a student who is unsatisfied with the allocation from the group and assign a different topic to the student, which

requires the involvement of academic staff. Creating a group where all students are satisfied with their allocation requires more knowledge of the application requiring more research.

9 References

1. Iwama, K.; Miyazaki, S.; Yanagisawa, H. Improved approximation bounds for the student-Project Allocation problem with preferences over projects. *J. Discreet. Algorithms* 2012, 13, 59–66.
2. Abraham, D.J.; Irving, R.W.; Manlove, D.F. Two algorithms for the student-Project Allocation problem. *J. Discreet. Algorithms* 2007, 5, 73–90.
3. Kuh, G.D. The national survey of student engagement: Conceptual and empirical foundations. *New Direct. Inst. Res.* 2009, 141, 5–20.
4. Teo, C.Y.; Ho, D.J. A Systematic Approach to the Implementation of Final Year Project in an Electrical Engineering Undergraduate Course. *IEEE Trans. Educ.* 1998, 41, 25–30.
5. Pudaruth, S.; Bhugowandeen, M.; Beepur, V. Multi-Objective Approach for the Project Allocation Problem. *Int. J. Comput. Appl.* 2013, 69, 26–30.
6. Anwar, A.A.; Bahaj, A.S. Student project allocation using integer programming. *IEEE Trans. Educ.* 2003, 46, 359–367.
7. Irving, R. W., & Manlove, D. F. (2007). Approximation algorithms for hard variants of the stable marriage and hospitals/residents problems. *Journal of Combinatorial Optimization*, 16(3), 279–292. <https://doi.org/10.1007/s10878-007-9133-x>
8. Algorithms for the Assignment and Transportation Problems. *Journal of the Society for Industrial and Applied Mathematics*, 5(1), 32–38.
9. R.M. Karp, U.V. Vazirani and V.V. Vazirani, An optimal algorithm for on-line bipartite matching, in: *Proc. STOC Conf* (1990) 352-358.
10. Vlaanderen K., Jansen S., Brinkkemper S., Jaspers E.
The agile requirements refinery: Applying SCRUM principles to software product management. *Inf. Softw. Technol.*, 53 (1) (2011), pp. 58-70, 10.1016/j.infsof.2010.08.004
11. Moxley RT 3rd. Functional testing. *Muscle & Nerve*. 1990 ;13 Suppl:S26-9. DOI: 10.1002/mus.880131309. PMID: 2233879.
12. Mithilesh Tarkar and Ameya Parkar. APIs and Restful APIs. *International journal of Trend in Scientific Research and Development (IJTSRD)*,
13. Atlassian. (2019). *Agile Best Practices and Tutorials* | Atlassian. Atlassian. <https://www.atlassian.com/agile>

14. *Documentation*. (n.d.). Node.js. <https://nodejs.org/en/docs>
15. *Getting Started – React*. (n.d.). Reactjs.org. <https://legacy.reactjs.org/docs/getting-started.html>
16. *HTML doctype declaration*. (n.d.). Wwww.w3schools.com. https://www.w3schools.com/tags/tag_doctype.asp
17. Mozilla. (2019, June 26). *CSS: Cascading Style Sheets*. MDN Web Docs. <https://developer.mozilla.org/en-US/docs/Web/CSS>
18. MySQL. (2019). *MySQL Documentation*. Mysql.com. <https://dev.mysql.com/doc/>
19. OpenJS Foundation. (2017). *Express - Node.js web application framework*. Expressjs.com. <https://expressjs.com/>
20. *Postman API documentation*. (n.d.). Postman Learning Center. Retrieved September 7, 2023, from <https://learning.postman.com/docs/developer/postman-api/intro-api/>
21. *React Components - Material UI*. (n.d.). Mui.com. <https://mui.com/material-ui/>
22. *Sequelize v6 | Sequelize*. (n.d.). Sequelize.org. <https://sequelize.org/docs/v6/>
23. Visual Studio Code. (2023). *Documentation for Visual Studio Code*. Code.visualstudio.com. <https://code.visualstudio.com/docs>
24. contributors, M. O., Jacob Thornton, and Bootstrap. (n.d.). *Get started with Bootstrap*. Getbootstrap.com. <https://getbootstrap.com/docs/5.3/getting-started/introduction/>