Anglia Ruskin University

MINGLEMATES: Crafting Cohesive Living Spaces through Intelligent Roommate Matching

Postgraduate Major Project

MOD002726

Megha Prasannan Pillai

SID: 2160346

Submitted: 26th January 2024

DECLARATION: I hereby confirm that this research is my own work and has been submitted to Anglia Ruskin University as part of the degree requirement. I affirm that the content presented in this dissertation complies with academic regulations and ethical standards. Proper citation has been employed for any information drawn from external references.

ACKNOWLEDGEMENT

I would like to express my gratitude to Mrs. Priyanka Sharma, Professional Skills Tutor at the School of Computing and Information Science, Anglia Ruskin University, Cambridge, my project supervisor. Mrs. Priyanka Sharma valuable guidance and encouragement significantly contributed to the successful culmination of this project. I am also thankful to Dr. Cristina Luca for her support throughout the research process. Finally, I extend my gratitude to all individuals who played a crucial role in the completion of this project.

TABLE OF CONTENTS

ABSTRACT

Embarking on the quest for compatible roommates and shared living arrangements, Minglemates is a cutting-edge platform that is purposefully made to lessen the common difficulties in this complex procedure. The program places emphasis on security, accessibility, and user involvement by using an intelligent flat mate matching algorithm, a transparent feedback and rating system, and efficient technological tools. The application's stability is ensured by comprehensive testing, which includes design validations, API testing, functionality testing, and feature testing. Minglemates excels in promoting streamlined cooperation among roommates, providing impactful technological solutions, and maintaining a dedication to security and ease of use. Although recognizing specific constraints and topics that require more exploration, such as geographical factors and sophisticated functionalities, the project provides significant perspectives to the field of research. This abstract provides a concise overview of the project's goals, design, execution, assessment, and its expected influence on the future direction of research and development in the field of flat mate matching platforms.

1. INTRODUCTION

Room sharing is crucial in tackling current societal and economic concerns, providing many advantages beyond just sharing costs. Amidst a time characterized by increasing housing expenses and economic instabilities, room sharing arises as a viable remedy, enabling individuals to combine their resources and alleviate financial hardships. Additionally, it promotes a feeling of trust and fellowship, mitigating the solitude that may come with living alone. In addition to its economic benefits, room sharing enhances sustainability by maximizing the utilization of available living areas. By using shared lodgings, the total demand for housing is reduced, leading to a more effective allocation of resources and a lesser environmental impact.

The cultural and sociological ramifications of sharing rooms are equally substantial. It enables various interpersonal relationships, dismantling societal obstacles and promoting cross-cultural interactions. This approach of linked living fosters mutual comprehension and acceptance, hence promoting the development of inclusive and harmonious living situations. Room sharing essentially functions as a practical solution to the difficulties presented by modern living situations. The inherent ability of this component to enhance economic efficiency, sustainability, and social cohesion makes it crucial for meeting the changing requirements of individuals and communities in the 21st century.

1.1 AIMS

The project is focused on developing a platform that streamlines the process of locating compatible housemates. The initiative attempts to simplify the intricate process of pairing persons with similar interests and finding them suitable living arrangements by utilizing sophisticated algorithms and data analysis that prioritize user preferences.

Furthermore, the initiative places a high priority on improving the entire user experience, in addition to providing capabilities for finding roommates. The goal is to guarantee that every engagement inside the platform is efficient, captivating, and pleasurable. This entails scrupulous creation of functionality and design components to conform to user expectations.

The company also aims to foster long-lasting relationships among roommates, which go beyond the limitations of the internet platform. The platform aims to facilitate the establishment of significant connections among individuals by integrating communication, comprehension, and cooperation elements.

Moreover, the project seeks to provide a flexible platform that can adjust to the varied requirements and preferences of its users. The website aims to provide an open and friendly atmosphere for all users, regardless of their career, student status, or personal lifestyle choices.

1.2 OBJECTIVES

- Matching Algorithm: The main goal is to create and apply a sophisticated matching algorithm that produces highly accurate roommate matches. This system utilizes a range of factors, such as lifestyle choices, personal interests, and room preferences, to determine compatibility rates that exceed traditional measures.
- Creating an intuitive user interface (UI) that effectively blends visual attractiveness with easy-to-use navigation is crucial for the product's success. The objective is to create a design that represents modern style while guaranteeing a smooth experience, enabling effortless navigation for users on the platform.
- The primary objective is to integrate real-time communication components into the platform. The application should provide users with the ability to communicate with potential roommates through messaging, allowing for effective and significant interactions.
- Integrating favorable and unfavorable components: The effectiveness of the roommate selection procedure is greatly dependent on user engagement. One particular goal is to offer features that enable users to indicate their choices by expressing their likes. Users may utilize this feature to efficiently refine their matches, hence enhancing the level of customization.
- Ensuring Data Security and Privacy: Implementing data security is essential due to the delicate nature of personal information. The platform aims to cultivate consumer confidence by placing a high priority on privacy and data security, ensuring the protection of their data.
- Executing user testing is a consistent goal throughout the development process. The platform will conform to user expectations and preferences by gathering user feedback and implementing pertinent improvements.
- The aims focus on creating a flat mate matching tool that simplifies the process of matching compatible housemates, for a satisfying and harmonious shared living experience. Explicitly established objectives and targets jointly contribute to the larger aim of changing communal living circumstances.

Ethics training was completed before the initiation of the research, and the required ethics form for this research project was submitted. The Ethics approval code assigned to this study is ‘ETH2324-0778' The research has been categorized as "GREEN," indicating that ethical approval is not deemed necessary for its execution.

2. LITERATURE REVIEW

Why is a perfect roommate required?

Zana (2020) highlights the crucial significance of roommates while moving to a foreign nation for educational or professional purposes. The text starts by emphasizing the initial obstacles individuals encounter when searching for appropriate housing, expressing apprehensions over the location, affordability, and credibility of adverts. According to the author, making an initial reservation with Airbnb might offer a pragmatic approach to familiarize oneself with a new location, comprehend transit networks, and directly assess possible housemates.

The author recounts personal anecdotes of experiencing insecurity and pressure before departure, arguing for the proactive search of an apartment beforehand as a means to lessen stress. The story emphasizes the importance of cohabiting with people in an unfamiliar environment, as it not only imparts vital life skills but also fosters personal development.

The author outlines five primary advantages of cohabitating with roommates:

1. Enrichment: The wide range of interests, narratives, and fervors among roommates expands perspectives, sparks inspiration, and has a good impact on one's character.
2. Sharing, whether it is food, goods, or support, promotes a sense of community and empathy that goes beyond personal need.
3. Experiencing a multicultural environment allows individuals to encounter diverse ethnic customs, festivities, and rituals, providing a distinct outlook on the global community's variety.
4. Collaborative activities, such as organizing trips or learning together, are more pleasurable and successful when done as a team.
5. Humor Balance: Roommates enhance the overall ambiance by engaging in tiny acts of kindness, such as exchanging morning greetings or offering useful suggestions, therefore fostering a motivated and intellectually challenging setting.

The article: "*Pros and cons of living with a roommate*" (Freeze, 2023) explores benefits of having a roommate, The advantages emphasized are:

1. Cost-effective Savings: Engaging in a roommate arrangement to divide living expenses, including rent, utilities, and other outlays, is a practical approach for renters to

accumulate financial resources. This layout makes it easier to live in a desired area or have a larger living space.

2. Economical Furnishing: Living with a roommate enables the shared payment of furniture costs in common areas, simplifying the task of furnishing a home at a reduced expenditure, as the financial burden is split among the residents.
3. Shared Responsibilities: Roommates collaborate to provide a fair allocation of domestic chores and duties, promoting a cooperative attitude to responsibilities such as cleaning and cooking. This joint endeavor results in increased efficiency in terms of both time and effort for all parties involved.
4. Living with a roommate not only offers company, but also cultivates a feeling of community. Participating in discussions, dining together, and discovering novel pursuits may be immensely pleasurable, particularly while adapting to a strange locale.
5. The inclusion of a roommate enhances the impression of safety and security in the residence. The presence of another individual provides a sense of security in the event of unexpected situations or emergencies.

Similar Applications

Roommates.com

In this application the procedure starts with the establishment of an individualized profile, enabling users to promptly upload photographs, furnish pertinent information, and finalize their profiles within a few minutes, underscoring the importance of presenting a favorable initial impression. After creating a profile, individuals are then requested to specify their preferences by responding to questions pertaining to their personal information and the area they plan to share. This data is utilized to ascertain appropriate matches for the user. Ultimately, the platform presents these matches to the user, providing a user-friendly and reliable means of connecting via the integrated messaging system. This tripartite methodology guarantees a smooth and effective encounter for users trying to establish connections with possible mates. (Roommates.com, LLC, 2023)

Spare Room

In Spare room user can create a persuasive advertising to establish a connection with prospective roommates by providing specific information about the availability of your room or your quest for a living place. Specify the desired nature of your flat share experience, whether it entails regular social interactions or infrequent get-togethers. Improve the

effectiveness of your advertisement by using images and a video to offer a realistic portrayal of yourself or your living environment. Discover accessible shared accommodations by doing targeted searches for desired places, refining outcomes, and employing sophisticated search features to match your tastes and financial constraints. Upon identifying possible matches, go to register in order to establish direct contact with them. If you find them intriguing, plan a meeting to meet face-to-face. Enhance the prominence of your advertisement by making a modest payment for improved search positioning or early access to recently posted shared accommodations. Utilize the platform to not only communicate practical information but also personal parts of your lifestyle, promoting ties that go beyond just living arrangements. (SpareRoom, 2023)

Front end frameworks for Web Application Development

The front end serves as the user interface layer of your application, encompassing everything that users interact with. While commonly referred to as the visible components of an application, it extends beyond that to encompass any code responsible for presenting data efficiently. Essentially, the front end entails the creation of user-friendly interfaces, along with the effective management, display, and real-time updating of data received from the back end or Application Programming Interface (API).

A front-end framework, in turn, serves as a foundational structure for constructing the user interface. Typically, it provides a framework for organizing files, incorporating features like components or a CSS preprocessor.

Front-end frameworks optimize the development process by tackling prevalent obstacles:

- Enhance maintainability by ensuring that the code is easily comprehensible, capable of being tested, and adaptive.
- Enhance the ability to create and reuse frequently used user interface components.
- Optimize DOM manipulations to minimize updates, resulting in smoother user interactions.
- Data-Driven UI Manipulation: Facilitate the modification of user interface elements in response to data inputs.
- Ensure consistency and intuitiveness by standardizing user interface components to create a unified and seamless experience.
- Minimizing Boilerplate Code: Remove redundant code to enhance efficiency.

- Common Language: Establishes a unified structure to facilitate smooth collaboration among developers.

Front-end frameworks are crucial for constructing intricate user interfaces with efficacy. They promote the development of modular and maintainable designs, which are backed by active communities and comprehensive documentation. These frameworks tackle issues related to scalability, make use of contemporary JavaScript, and offer tools for prototyping, promoting a common language for efficient communication among developers (Pekarsky, 2020)

Frontend frameworks are sets of pre-written code and tools that aid in the UI/UX design and development process for websites and online applications. These frameworks give web developers an organized method with a collection of reusable parts, libraries, and tools to make the process go more quickly (Dhaduk, 2024).

React JS

ReactJS is a freely available library specifically created for building user interfaces, particularly for single-page apps. Developers are enabled to design robust web apps that can effectively handle data and update in real-time without the need for a page refresh. React utilizes a sophisticated diffing method, enabling it to selectively extract only the essential data from the Document Object Model (DOM) while leaving everything else unaffected. Utilizing reusable components in React streamlines application development, promoting a straightforward approach to organizing applications. The intelligent architecture of React guarantees a uniform user experience, freeing programmers from superfluous responsibilities and allowing them to concentrate on more crucial capabilities and business logic. Significantly, React does not enforce a particular methodology for tasks, providing a diverse range of libraries that users can select from to execute specified operations. In addition, React has crucial functionalities such as lifecycle methods and React Hooks, which govern the sequence of events that take place during the lifespan of a component. This article examines many facets of the ReactJS framework, emphasizing the ways in which these characteristics enhance the process of building applications. Furthermore, it explores frequently employed patterns, their practical uses, and the methods for incorporating them into our programs. (Rawat et al., 2020).

React components are the essential elements used to construct user interfaces. Creating a complete user interface entails constructing a hierarchical arrangement of several components. The render() function of each React component is essential for building an intermediate Document Object Model (DOM). The intermediate Document Object Model (DOM) is

generated by a recursive procedure that begins by using the React.*renderComponent*() function on the main component. (Aggarwal, 2018)

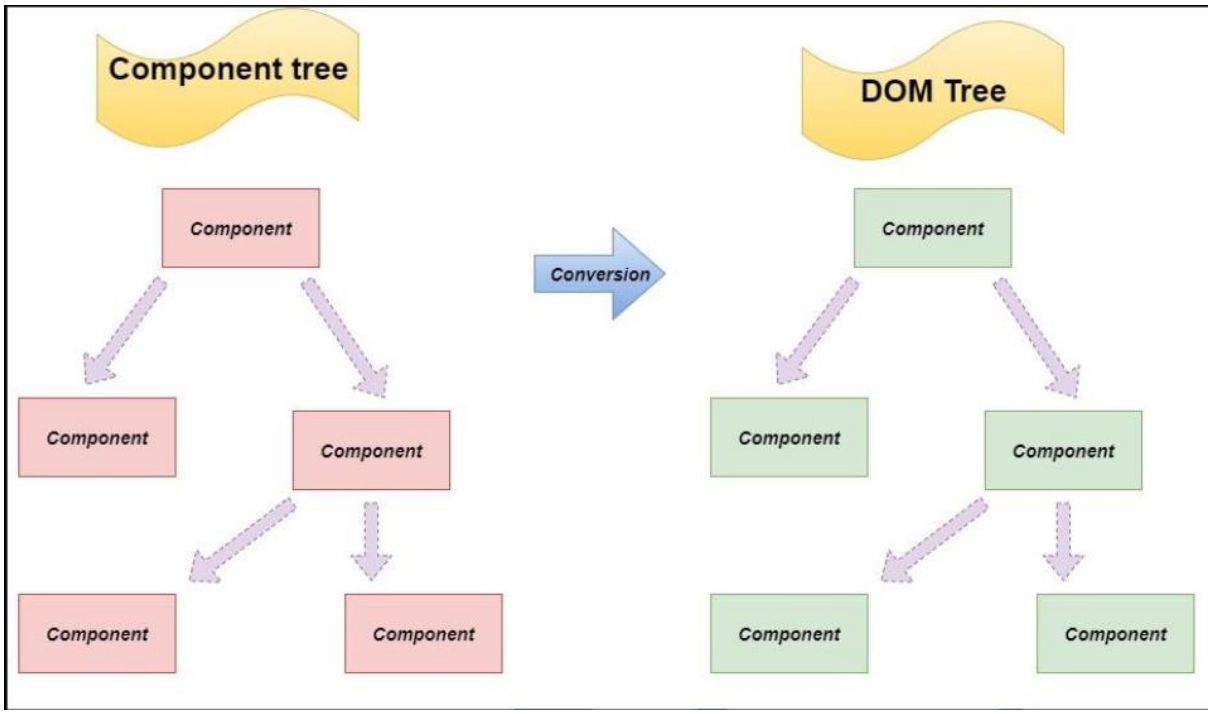


Figure1: Conversion to HTML DOM

(Source: Aggarwal, 2018)

The ReactJS framework collaborates JavaScript's speed and efficiency with an optimized approach to manipulating the DOM, resulting in accelerated web page rendering and the creation of extremely dynamic and responsive online applications. (Herbert, 2023)

Angular

Angular has become a prominent framework for building responsive single-page web apps and visually attractive components for websites, solidifying its position as one of the most widely used JavaScript frameworks. Angular was originally designed as a complete framework that prioritizes clarity and robustness in the development process. Developers value its remarkable functionalities, especially in the development of interactive single-page apps and its support for the Model-View-Controller (MVC) architecture. The significant increase in the adoption of Angular in corporate environments may be due to its highly responsive nature and its broad range of features. Unlike other frameworks that have difficulties with existing technologies,

Angular sets itself apart by employing thorough engineering practices to guarantee optimal performance and consistent, trustworthy results. (Geetha et al., 2022)

AngularJS is a front-end web application framework. One of the distinguishing features of Angular is its use of the standard Document Object Model (DOM) in conjunction with TypeScript, a statically typed superset of JavaScript, to improve the readability of code and maintainability. This framework adds two-way data binding to applications, resulting in a smooth interaction among the model (data) and display components. (Sanity, 2023). Angular is built on a component-based design and was created with TypeScript. Angular lets developers construct their own components. These components can be reused. Angular may be used for both unit and end-to-end testing of an application. (Cetin, 2023)

Vue.js

Vue is a JavaScript framework for creating user interfaces. It builds on basic HTML, CSS, and JavaScript to give an object- and component-based programming architecture for creating simple or sophisticated user interfaces. (Vue.js, 2023). Vue.js can have a favorable impact on the development process due to specific capabilities of the framework. Furthermore, the thesis serves as a guide for building a generic Vue.js project and offers readers with the author's coding standards and best practices, which are extremely useful for inexperienced developers. (Tran, 2020). Vue.js is generally used to create dynamic and attractive web interfaces. Vue has a feature called MVC, or Model View Controller architecture. This architecture allows developers to observe the user experience of their project, whether it is a mobile app, an application for the desktop, or a website. (Tuama, 2022).

Comparison of Front-end frameworks

Research done by Saks, E.(2019) gathered insights on framework popularity from prominent cloud service providers, including GitHub, NPM, and Stack Overflow. GitHub data indicated Vue's slight lead over React, while Angular showed lower stars and more issues. React significantly outperformed in NPM downloads, surpassing the combined downloads of the other two frameworks. While Angular may require fewer third-party libraries, Reacts dominance on NPM is evident. On Stack Overflow, Angular's popularity appears to have peaked, with React poised to surpass it. Both React and Angular contribute between 2.5% and 3% of total questions, while Vue, experiencing steady growth, constitutes only 8% of total questions asked.

Having reached a level of maturity and receiving significant contributions from the community, React has achieved broad acceptance and is considered a reliable choice among front-end JavaScript frameworks. The favorable job market for React suggests a promising outlook for its future. Ideal for newcomers to front-end development, startups, and developers valuing flexibility, Reacts capacity for seamless integration with other frameworks positions it as a strong contender. The framework's established reputation and adaptability make it a preferred option in the ever-evolving realm of front-end development. (Daityari, 2023)

The data from npm trends (Figure 2) clearly demonstrates React enduring and unequivocal conclusive in popularity among JavaScript frameworks. React constantly demonstrates a greater number of downloads in comparison to Angular and Vue, indicating its extensive adoption and use in practical projects. The continuous relevance and strong community support of React highlight its appeal to developers and organizations looking for a powerful and widely-used front-end framework to create modern online apps.

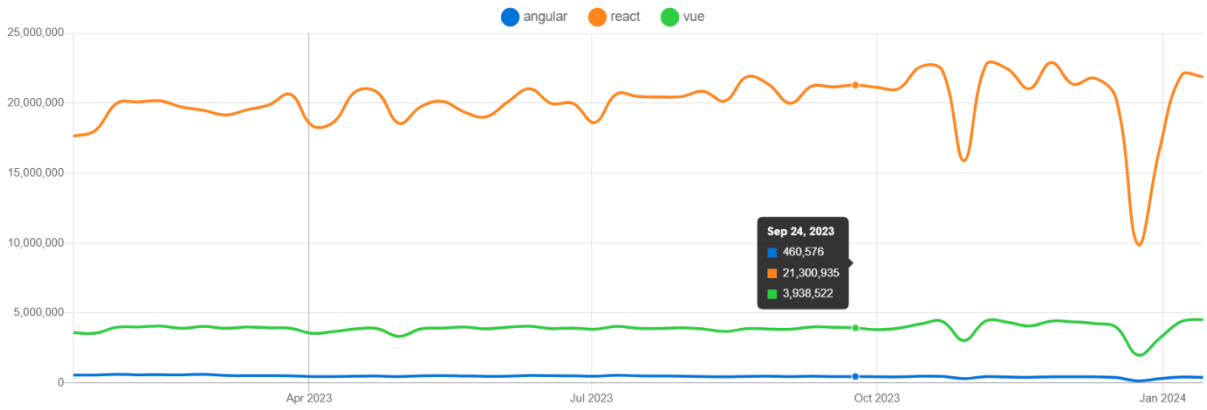


Figure:2 NPM Trends

Source: (NPM trends, 2023)

Summary of Comparison

The literature surrounding reusable components highlights the significant role that React plays in streamlining the process of application development and organization. The research appropriately recognizes the versatility of React, which enables developers to select from a diverse range of frameworks without enforcing a certain technique. Studying React components and their use in building user interfaces offers valuable understanding of the framework's structure.

Contrasting React with other front-end frameworks like Angular and Vue.js provides readers with a more comprehensive understanding of React advantages, notably in terms of its

widespread usage, strong community backing, and flexibility. The incorporation of React's functionalities and its synergy with JavaScript's optimization help to expedited web page rendering, bolstering the creation of dynamic and responsive online apps.

In addition, the study expands its analysis to encompass NPM patterns, showcasing React's persistent popularity and extensive integration in real-world applications. An in-depth analysis of the popularity of frameworks, using data from GitHub, NPM, and Stack Overflow, offers important insights into the preferences and trends among developers.

Overall, the research provides a thorough and detailed resource for developers and organizations who are contemplating the use of ReactJS. This book not only provides a detailed overview of the framework's features and capabilities, but also situates React within the wider context of front-end development. As a result, it proves to be a significant asset for anyone navigating the constantly changing field of web development.

Java for web application development

The majority of Java libraries, which are mostly free and open-source, provide professional assistance, greatly expediting the backend development of web applications. Java provides a wide range of libraries that cater to various purposes, including logging, JSON parsing, unit testing, XML and HTML parsing, communications, PDF and Excel reading, encryption, and more. Java's efficiency and accessibility for web programming are attributed to its simplicity and ease of creation, learning, maintenance, and debugging, as well as the broad support of open-source libraries. Java outperforms Node.js in several dimensions, rendering it a favored option for some backend development scenarios. Java's merits reside in its statically-typed nature, which results in improved performance and stability. The robustness of Java is enhanced by security features such as the Java Virtual Machine (JVM) which provides memory protection and isolation. Moreover, Java possesses a wide range of libraries and tools, which greatly aid in the creation and upkeep of complex applications. (Fagbuyiro , 2023)

The Spring Boot framework is widely used due to its numerous benefits in application development. It includes functionalities such as MVC support and the availability of RESTful Web Services. Moreover, Spring incorporates inherent assistance for establishing database connections within its package. An outstanding advantage of the Spring framework is its support for dependency injection, which simplifies the setting of dependencies in the application development process for enhanced ease. One major advantage of using Spring Boot is the effortless integration with a Tomcat server, enabling easy inclusion and direct execution.

This enhances the efficiency and user-friendliness of application development and deployment. Spring Boot is widely recognized for its extensive range of capabilities, such as MVC, RESTful Web Services, database connection, and strong support for dependency injection, which makes it a favored option among developers. (Suryotrisongko et al., 2017)

Summary of Literature Review

The literature review starts by emphasizing the importance of having an ideal roommate, particularly when relocating to a foreign country for scholastic or professional reasons. The text highlights the early difficulties of locating appropriate accommodations and the benefits of platforms such as Airbnb in facilitating the process. The advantages of living together with roommates are delineated, including personal growth, economical savings, collective obligations, and a feeling of togetherness.

The significance of front-end frameworks in the development of user-friendly interfaces is underscored, with an emphasis on their contribution to improving maintainability, component reusability, optimizing DOM operations, and facilitating a shared language among developers. ReactJS, Angular, and Vue.js are renowned front-end frameworks, each distinguished by its distinct characteristics and skills. The evaluation of these front-end frameworks incorporates analysis from GitHub, NPM, and Stack Overflow, emphasizing React's widespread usage, adaptability, and robust community backing. The research finishes by providing a concise overview of the benefits of React and its enduring prevalence in practical implementations.

The literature also discusses online sites such as Roommates.com and SpareRoom, which enable individuals to seek for roommates. These platforms offer personalized profiles and use pairing algorithms that consider individual preferences.

Finally, the review address Java's involvement in the building of online applications, highlighting its effectiveness, comprehensive library support, and superiority over Node.js in specific situations. The Spring Boot framework is highly acclaimed for its notable attributes, which encompass MVC support, RESTful Web Services, database connectivity, and effortless integration with a Tomcat server.

Research Gap

The literature study offers a thorough examination of roommate concerns, front-end frameworks, and Java's significance in web development. However, there are several areas that warrant further investigation:

- User Perspectives on Roommate Experiences: Although the research emphasizes the benefits of having roommates, its main emphasis is on the pragmatic elements. Integrating qualitative data or surveys to capture the subjective experiences and obstacles encountered by persons cohabitating with roommates might yield a more detailed comprehension.
- A comparative analysis of front-end frameworks, such as React, Angular, and Vue.js, is available in the literature. However, a more comprehensive investigation is needed to explore particular use cases, performance benchmarks, and real-world applications for each framework. This might provide developers with guidance in selecting the best appropriate framework for their applications.
- The literature examines front-end frameworks and Java as independent entities for web development, however there is a need to integrate these technologies. Examining the most effective methods for smooth integration might yield a comprehensive comprehension of web application development.

By addressing these research gaps, this project can enhance understanding of roommate matching, front-end frameworks, and the incorporation of technology in online application development for this roommate matching platform.

3. DESIGN OF THE PROPOSED WEB-APPLICATION

The design section is the foundation on which the complete Minglemates is developed. This crucial stage involves a sequence of tightly linked elements that together determine the user experience and functioning of the application. Each stage in the design process, from producing ideas for modules to be developed and creating wireframe representations to choosing the most suitable tools and technologies, was essential in determining the overall structure. This section further delves into detailed exploration of the design, offering a thorough overview that covers the entire process from early planning to the visualization of user journeys, the design of the compatibility calculation algorithm, and the database design. Let's explore the carefully designed sections that create a user-centric Roommate Matching Platform.

3.1 PROPOSED FUNCTIONALITIES FOR IMPLEMENTATION

The Minglemates application is a integration of many crucial modules necessary to accomplish its goals and guarantee a smooth user experience:

- **Registration Module:** enables new users to join the platform by filling out a detailed registration form. The form gathers crucial data required for the establishment of an account.
- **Login Module**: The login feature enables users to safely log into their accounts by entering their registered email and password. This module improves the security and tailored experience for every user.
- **Profile Module**: The purpose of the Profile Module is to allow users to customize their profiles according to their preferences. Users have the ability to add a profile photo, offer a brief biography, and establish gender preferences. The module also enables the storing and alteration of personal information such as age, gender, and employment, enabling users to provide a comprehensive and precise depiction of themselves.
- **Preferences Module**: This module is crucial in influencing the user experience as it enables consumers to clarify their lifestyle choices. Users have the ability to generate a compilation of essential lifestyle details that are necessary for determining compatibility. Furthermore, consumers have the ability to customize their room preferences by selecting from a variety of alternatives. The module offers flexibility by allowing users to modify these options as their needs change.
- **Matching Roommates Module**: This module is the core of the application containing the matching algorithm. The algorithm computes the level of compatibility between

users by using the lifestyle choices contained in the choices Module. This approach serves as the basis for creating roommate matches, guaranteeing that users are linked with persons who have similar interests and values, promoting pleasant living arrangements.

- **Messaging Module:** The Messaging Module enables efficient communication. Users have the ability to exchange messages, which promotes significant exchanges and facilitates a better understanding of possible housemates. This module improves the overall user experience by offering a platform for smooth and uninterrupted communication.
- **Feedback Module:** User feedback and rating is highly regarded and collected through the Feedback Module. Users could utilize this feature to communicate their experiences, comments, and thoughts regarding the platform. User input is crucial in driving continuous enhancements and guaranteeing that the platform adapts to fulfil user expectations.
- **Testimony Module:** The Testimony Module functions as a platform to display favorable user experiences. Users are given the chance to offer testimonials on their experience utilizing the site. These testimonials enhance confidence and authenticity by demonstrating the platform's efficacy in connecting matching roommate.
- **Logout Module**: The Logout Module guarantees a safe and user-friendly termination of the user's session within the Minglemates application. This module provides users with the ability to log out of their accounts, presenting a concise procedure to terminate their current session.

These interrelated modules work together to create a strong and user-focused Minglemates application, by supporting the primary goal of facilitating effective and harmonious roommate matching.

3.2WIREFRAMES

A wireframe acts as a diagram or plan, assisting in the facilitation of communication and thought processes among developers, designers, and project stakeholders. This section explains the wireframes created at the project inception, which are used as a foundational stage to visualize the structure of the application during the development process.

User Registration page

The registration page(Figure 4) should be designed to encompass the requisite form for creating a user account within the application. The mandatory fields include first name, last name, email id, password, and confirm password. Subsequent to these fields, a submission button is incorporated to facilitate the completion of the registration process. Figure shows the wireframe designed for the registration page.

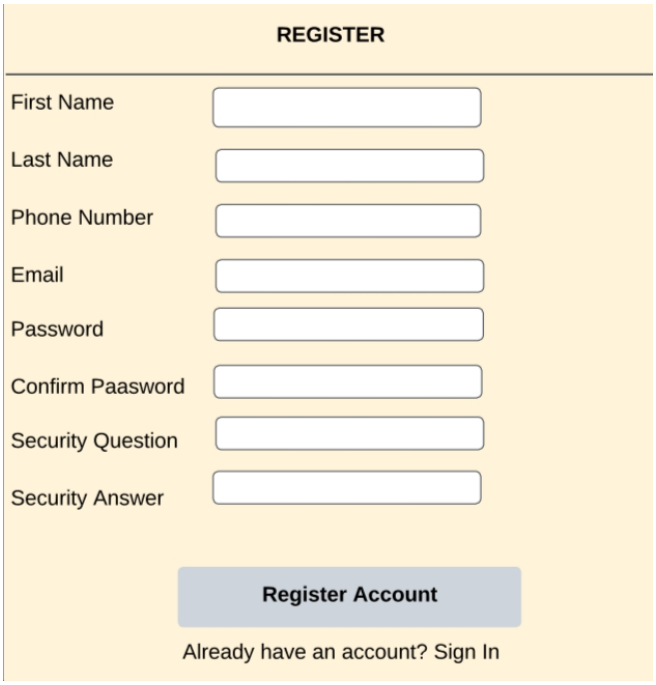


Figure 3: Registration page Wireframe

Login Page

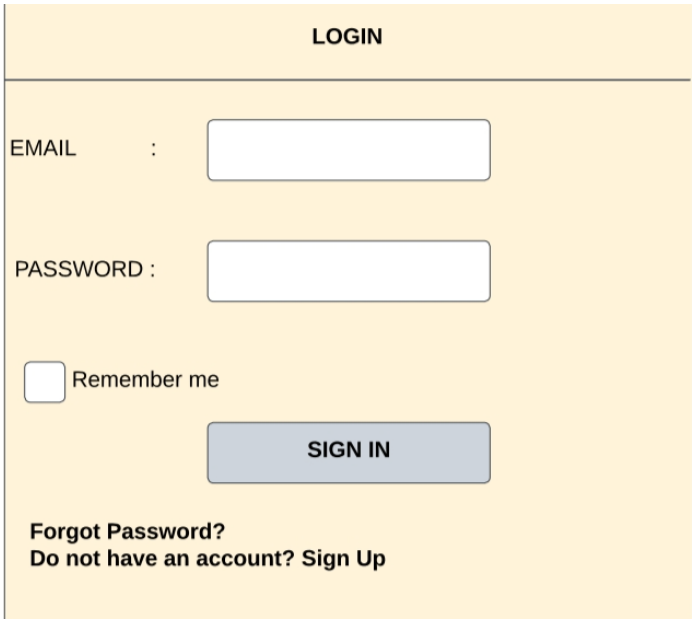


Figure 4: Login Wireframe

The registration page should be designed to encompass the requisite form for filling the credentials required for signing the application. The fields include email id and password. Subsequent to these fields, a submission button is incorporated to facilitate the login process. (Figure4)shows the wireframe designed for the login page.

Profile Setup page

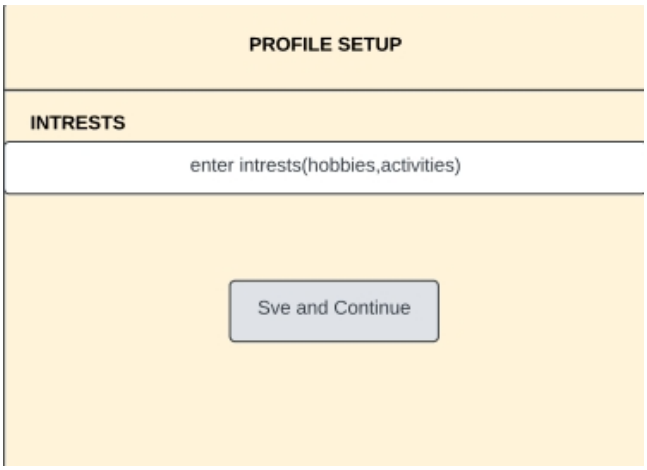


Figure 5: Profile setup Wireframe

The profile setup page(Figure5) enables users to set fundamental personal information, including components such as profile images, gender orientation, and interests. The interface should be created with a user-friendly style that includes a dedicated button for effortless storing of the preset details into the underlying database. The wireframe, illustrated in the figure, clearly portrays the user-friendly design for the page for easy interaction and entry of profile information.

Personal Information page

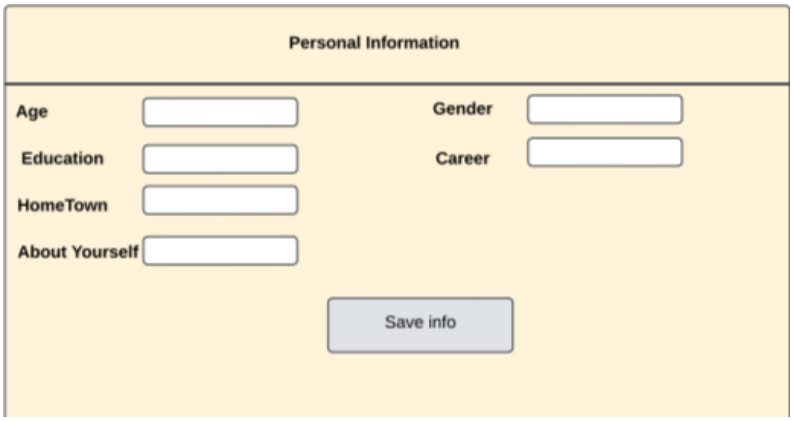


Figure 6: Personal Information Wireframe

Personal information page(Figure6) should be developed to add/update the personal information of a user sch as age, gender, education, career, home town, and other details about themselves. Subsequent to these fields, a submission button is incorporated to save these information to the database. The wireframe designed for this page is as illustrated in figure.

Preferences Setup page

The purpose for developing the Preference setup page(Figure7) is to facilitate the process of adding and editing user-specific preferences. These preferences include important fields such as the desired location, monthly budget, expected move-in date, and preferred gender of possible roommates. The wireframe, depicted in the figure, outlines the planned design used allowing users to efficiently engage and enter their particular preferences.

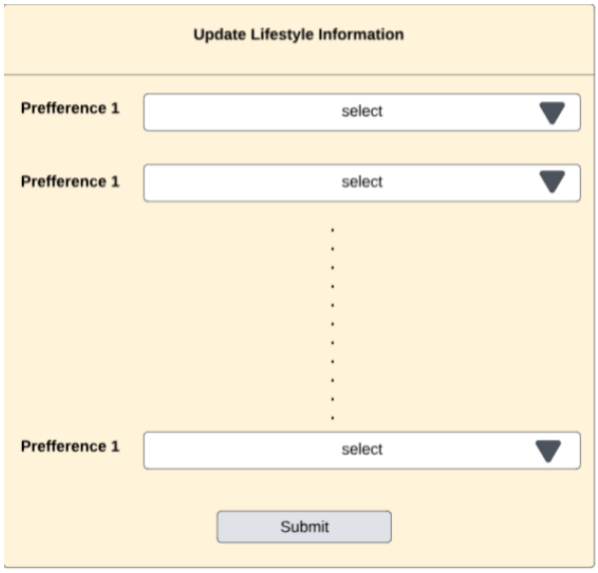


Figure 7: Preference setup page Wireframe

Room Selection Page

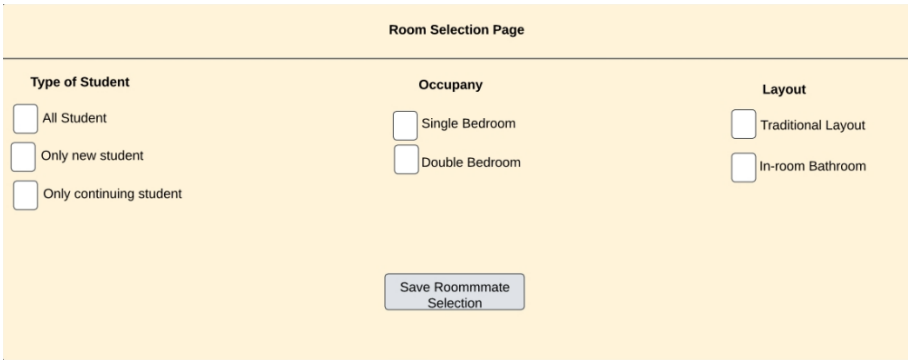


Figure 8: Room Selection page Wireframe

The Room Selection page(Figure8)is an essential feature that has to be developed in order to allow users to choose their preferred room combinations. This entails identifying the ideal characteristics of a housemate, stating preferences for either single or double occupancy, and choosing the desired room arrangement, such as the presence of en-suite amenities. The wireframe, seen in the figure, showcases the designed structure aimed to assist users in expressing their room selection preferences.

Dashboard/Home page

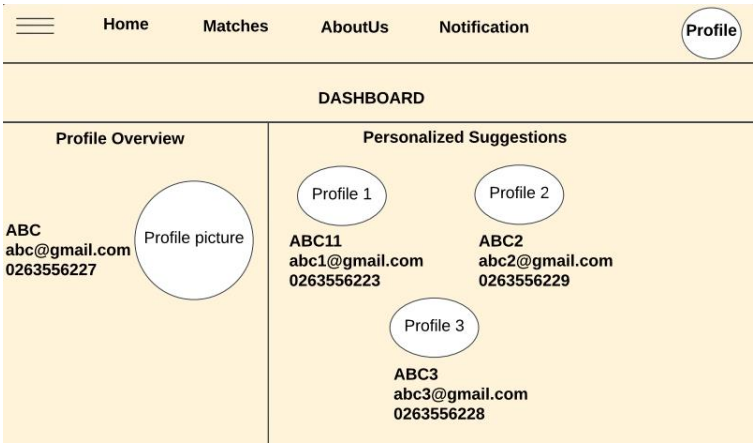


Figure 9: Dashboard/Home page Wireframe

The application's Dashboard (Figure9) should be designed to have a navigation bar that allows easy access to important pages such as the Home page, Matches page, About Us page, and Notifications. The navigation bar will prominently showcase the current user's name on the right-hand side. In addition, the Dashboard/Home page will have a sidebar that allows for easy access to the Feedback page, Testimonials page, Messaging page, and a user logout option.

The Dashboard/Home page should display overview of the user's information. Additionally, the page will display tailored roommate pairings for the user, along with a compatibility percentage. Users will have the ability to click on the profiles of suggested roommates, which will direct them to the selected roommate's profile page for a more in-depth viewing of their

profile. The wireframe, seen in the figure, vividly portrays the design and layout considerations included into the Dashboard/Home page.

Messaging Page

The Messaging Page (Figure10) of the application should provide users with an intuitive interface, facilitating smooth communication for message exchange. This particular feature serves as space where users can engage in conversations to enhance their understanding of each other. The wireframe, depicted in the accompanying picture, represents the deliberate design choices made to guarantee a seamless and user-friendly communications experience.

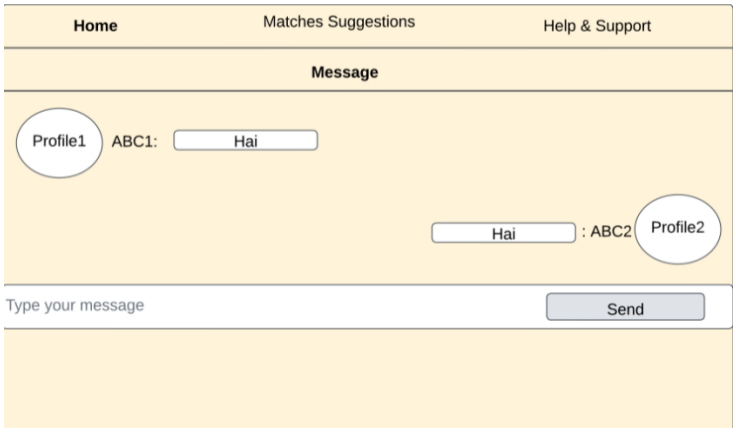


Figure 10: Messaging page Wireframe

Matching Suggestion Page

A page has to be developed to gather a comprehensive list of possible roommates, showcasing compatibility percentages alongside each user's profile. In order to enhance user engagement, each individual profile is equipped with a message option, enabling users to start communication with certain to-be roommates they find interesting. The wireframe, depicted in the figure11, illustrates the design developed for this page.

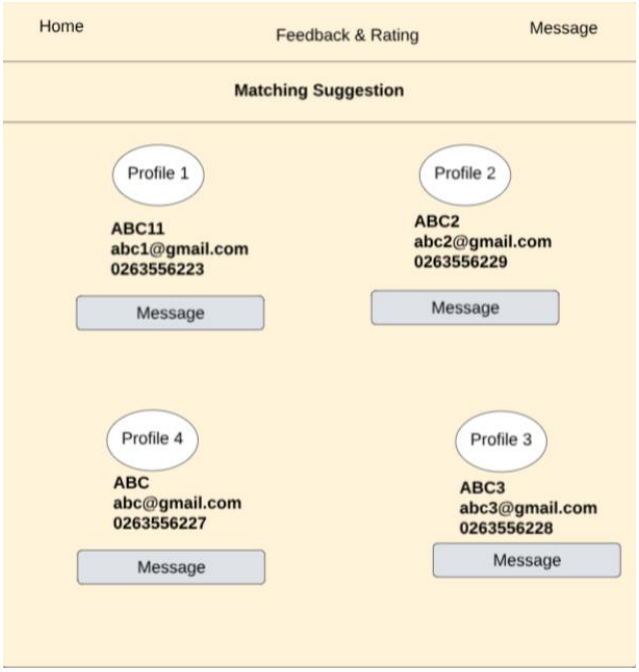


Figure 11: Matching Suggestion page Wireframe

Profile page

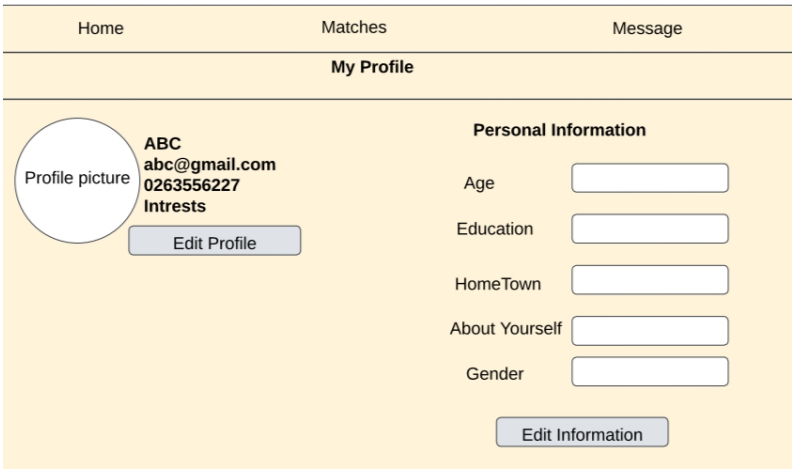


Figure 12: Profile page Wireframe

A dedicated webpage has been developed to provide a thorough showcase of all the profile information linked to the current user. Users are provided with the option to modify their information according to their requirements on this page. The left side of the interface showcases details provided by the user during the initial profile setup, while the right side showcases the personal information added by the user.. The wireframe, illustrated in the figure12, graphically portrays the design for this Profile page.

Testimonial page

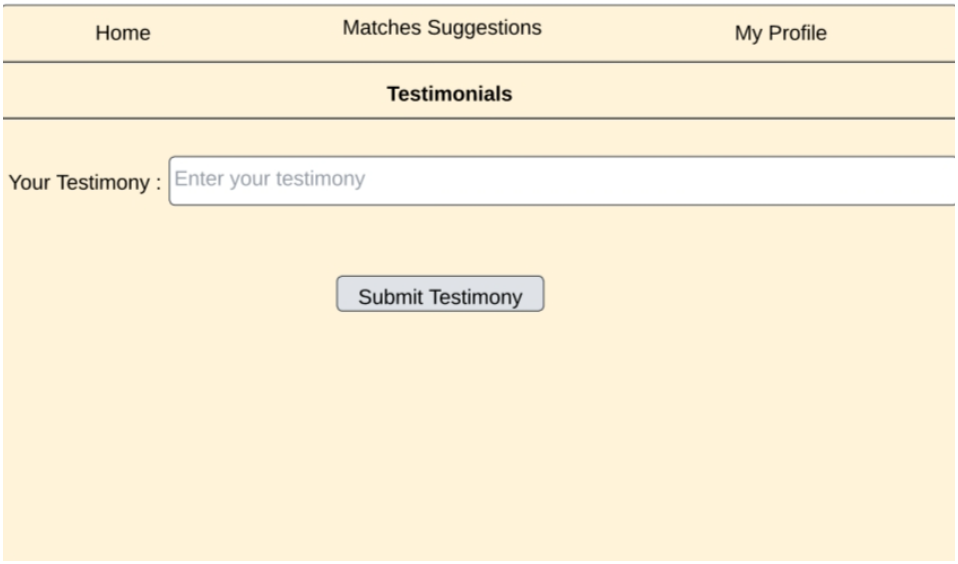


Figure 13: Testimonial page Wireframe

A Testimonial page (Figure13) has to be developed to enable users to express their opinions and recount their experiences related to the application. The wireframe, shown in the figure, demonstrates the design of this page. Users should offer testimonials, providing useful perspectives and contemplations on their engagements with the application. comments.

Feedback page

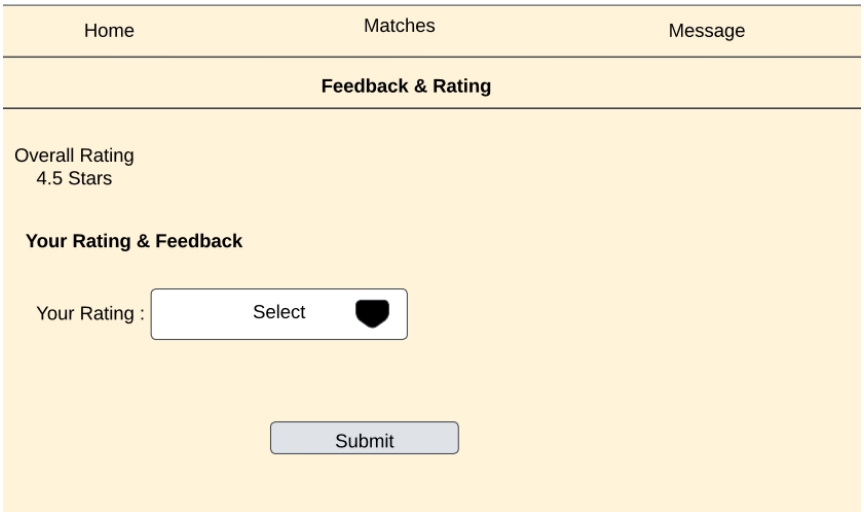


Figure 14: Feedback page Wireframe

A web page has been developed to assist users in submitting ratings and detailed feedback on the application. The wireframe, depicted in the illustration, provides a visual representation of the page layout of this feedback platform. In figure 14, Users should be able to both add and

modify ratings, which allows for flexibility and ensures that their changing opinions are accurately reflected. The design promotes user engagement and incorporates a system for ongoing enhancement through useful user feedback.

TOOLS AND TECHNOLOGIES USED

Technologies Used:

- Front End
    - React JS library
    - JavaScript
    - Tailwind CSS
    - HTML
- Back-end
    - Java 17
    - Spring Boot Framework
- Database
    - MySQL

Tools Used:

- Microsoft Visual Studio
- Spring Tool Suite
- XAMPP (for database)
- Talend API Tester

3.3 USE CASE DIAGRAM

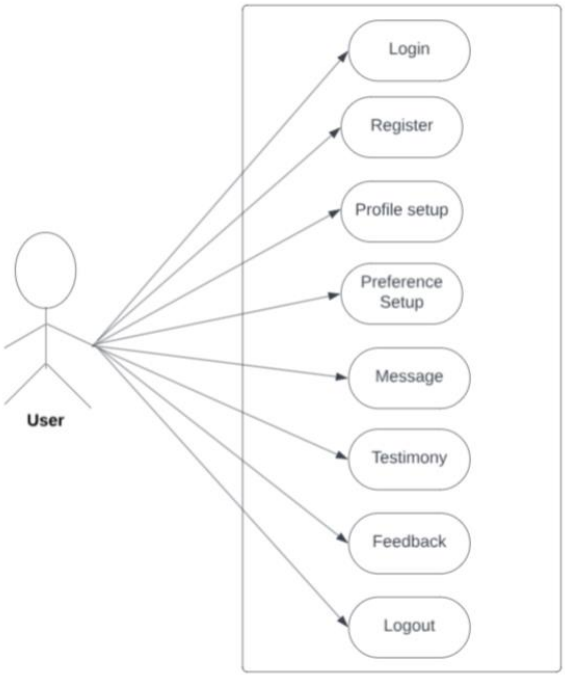


Figure 15: Use case diagram

This figure15 depicts a use case scenario, illustrating various interactions that a user may engage in with a given technology. The graphic features a stick figure denoted as "User" positioned on the left side, symbolizing the person engaging with the system. There are eight ovals that are connected to "User" by lines, each representing a distinct activity.

- Login: Existing users can securely access the web application by logging into their accounts.
- Register: New users should be able to create an account, facilitating their sign in into the application.
- Profile Setup: After registration, users can personalize their profiles, providing essential information to optimize their experience.
- Preference Setup: Users enjoy the flexibility of tailoring their preferences within the application, ensuring customized and relevant interaction.
- Message: The application enables users to engage in seamless communication by sending messages, fostering effective interaction between users.
- Testimony: Users should share their experiences and feedback through testimonials, contributing valuable insights to the web application's community.

- Feedback: An interactive feedback feature allows users to express their thoughts and opinions, providing a mechanism for continuous improvement based on user input
- Logout: To ensure security and privacy, users can conveniently log out of their accounts, concluding their sessions securely.

These features collectively form a comprehensive framework, promoting user engagement, customization, and feedback, thereby enhancing the overall utility and effectiveness of the application.

3.4 MVC FRAMEWORK

The MVC design enhances software development by partitioning issues into separate levels, hence introducing structure and organisation. The Model is responsible for managing data operations, the View is dedicated to user interface components, and the Controller serves as the coordinator, ensuring smooth communication between the Model and View levels. The modular and well-defined structure of this software project improves maintainability, scalability, and general efficiency in its development and management. (Sheldon, 2023)

3.5 DATABASE DESIGN

The "roommate_db" database (Figure16) is a relational database that functions as the foundation for a Roommate Matching application, with the goal of streamlining the process of locating suitable roommates. The database, built with MySQL, is designed to efficiently hold many forms of information on users, preferences, interactions, and testimonials. It consists of numerous tables, each specifically designed for a certain category of data. The architecture follows standardized standards, which ensure the integrity of data and enable easy querying.

User Information:

The database is built upon the "user" table, which stores essential user information. The table comprises columns labelled "id," "email," "first_name," "last_name," "password," and "phone_number." The "id" column functions as the main key, guaranteeing that each user possesses a distinct identification within the system.

Personal Information:

The "personal_information" table expands upon the user details and is connected to the "user" database by the foreign key "user_id". This table includes supplementary data such as

"about_yourself," "age," "career," "education," "gender," "hometown," and "relationship_status."

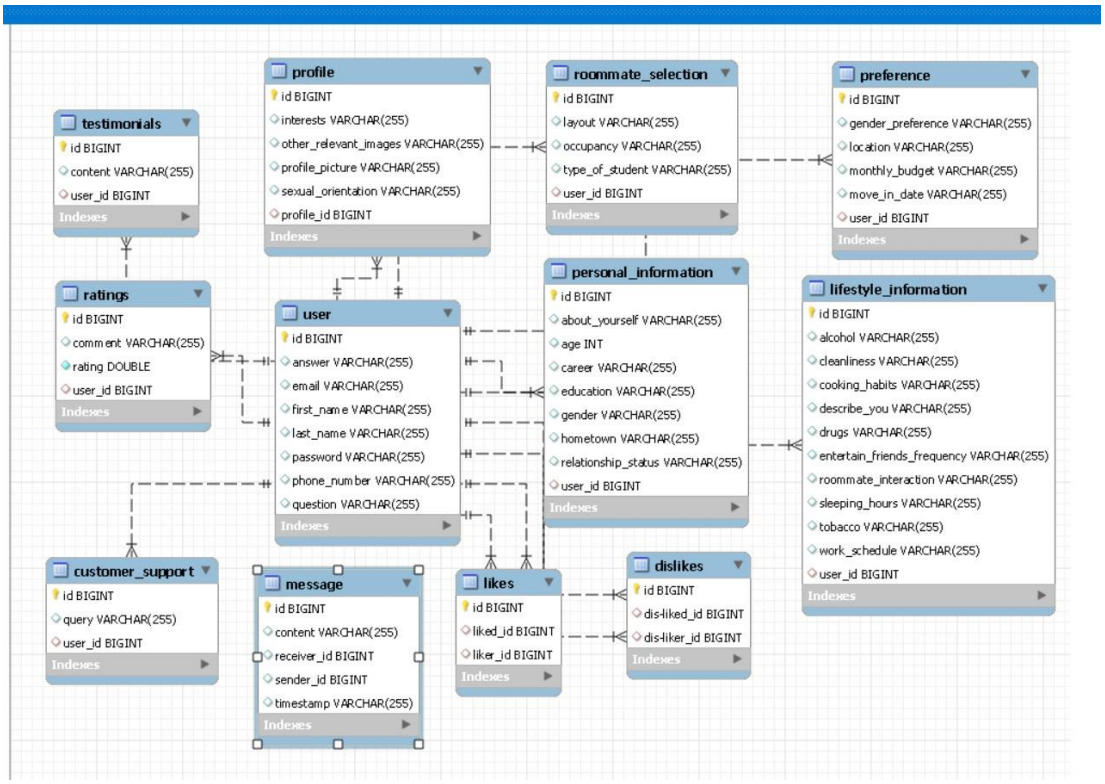


Figure 16: Database designed for the application

Lifestyle Information:

The "lifestyle_information" database is linked to the "user" table using the foreign key "user_id" to capture preferences related to lifestyle. The features covered include alcohol intake, hygiene practices, cooking habits, self-description, drug usage, frequency of entertaining friends, interaction with roommates, sleeping hours, tobacco usage, and job schedule.

Preferences:

The "preference" table, which is connected to the "user" database via the "user_id" foreign key, stores user preferences. The system stores data such as "gender preference," preferred "location," "monthly budget," and "move-in date" to assist the matching algorithm in generating appropriate roommate recommendations.

Roommate Selection Process:

The "roommate_selection" data is linked to the "user" table, specifically pertaining to the sort of roommate the user is looking for. Attributes such as "layout," "occupancy," and "type_of_student" help customise housemate recommendations to fulfil certain criteria.

Profile Likes/Dislike:

User interactions are controlled using the "likes" and "dislikes" tables. The "likes" table documents occurrences when a person expresses positive sentiment towards another user, and the "dislikes" table documents instances of negative sentiment. Both tables employ foreign keys to establish a connection with the "user" table.

Communication via messages:

The "message" table serves the purpose of enabling communication among users by storing messages along with specific information such as "content," "receiver_id," "sender_id," and "timestamp." The columns "sender_id" and "receiver_id" are associated with the "user" table.

Customer feedback :

User feedback is stored and shown in the "testimonials" and "ratings" databases. The "testimonials" table stores user testimonials, whereas the "ratings" table contains comments, ratings, and references to the "user" table.

User's Profile Information:

The "profile" table handles supplementary profile-related information such as "interests," "other_relevant_images," "profile_picture," and "sexual_orientation." The "profile_id" foreign key establishes a connection between the "user" table and this entity.

Customer Service:

The "customer_support" table is connected to the "user" database through the usage of a foreign key called "user_id" in order to manage customer support queries.

Ultimately, the "roommate_db" database serves as a thorough and efficiently structured foundation for the Minglemates application. The architecture of the system focuses on maintaining relational integrity, which guarantees efficient storage and retrieval of data, and establishes the groundwork for a smooth and successful roommate matching process.

3.6 DESIGNING APPLICATION USER JOURNEY

The Minglemates application guarantees a user-friendly and smooth experience by implementing a clearly defined user path. The primary phases of this trip encompass User Enrollment, User Login, Profile Setup, Preferences and Room Selection, Dashboard Interaction, Roommate Recommendations, Messaging, Notifications, Feedback and Ratings, Password Reset, Profile Analysis, and Logout.

- User Registration: The process begins with User Registration, during which new users submit necessary information to establish an account on the web application.
- User Sign-in: Registered users can authenticate themselves by logging in with their credentials, therefore obtaining access to the services and features offered by the platform.
- Profile Setup: After logging in, individuals are encouraged to establish their profiles by providing personal information, lifestyle preferences, and other pertinent characteristics.
- Preferences and Room Selection: Users click through pages to establish their preferences, including desired locations, monthly budgets, move-in dates, and orientation preferences. The room selection procedure enables customers to indicate their student category, desired occupancy, and layout preference.
- Dashboard: Upon finishing the profile setup and settings, users are automatically routed to the dashboard. Here, they are provided with recommendations for possible roommates using compatibility algorithms.
- Roommate Recommendations: The application produces roommate suggestions, presenting profiles and compatibility scores. Users have the ability to choose housemates based on their preferences or aversions.
- Messaging Component: Users interact with potential roommates through a messaging tool, enabling conversation and cooperation.
- Notification: Users are kept updated on recent activities by receiving alerts on the dashboard if there are new messages or matches.
- Feedback & Ratings: Users are given the opportunity to offer feedback, ratings, and testimonials on their experiences on the site.
- Password Reset: If users forget their password, they may use the password reset option to recover access to their accounts.

- Profile Analysis: Users may gain further insights on the proposed roommates by accessing their profiles, facilitating the process of making well-informed judgements.
- Logout: In order to guarantee secure entry to the platform, users have the option to log out of their accounts.

The flowchart visually illustrates the user's path throughout the Minglemates application, highlighting the sequential and interrelated structure of the application's capabilities. The design seeks to optimize the user experience and simplify the roommate matching procedure.

3.7 COMPATABLITY ALGORITHM

Roommate matching Algorithm:

Step 1: Fetch current user information (Preference, Roommate Selection, Personal Information, Lifestyle Information) from the database.

Step 2: for each user *Ui* from other users list

Step3: fetch user *Ui* information from the database

Step 4: calculate the compatibility of the user with current user by comparing different personal information (Preference, Roommate Selection, Personal Information, Lifestyle Information)

Step 5: Get the profile details of user Ui and add the details and compatibility calculated with the current user to an object.

Step 6: Append the object with user *Ui* information along with compatibility to a list of Compatible users.

Step 7: end of step 2 loop

Step 8: Return the list of Compatible users and their information.

Compatibility calculation:

$C=0+(N\times4)$

In this equation:

C is the compatibility score.

N is the number of criteria matched.

This formula signifies that the compatibility score starts at 0%, and for each criterion matched, the compatibility increases by 4%. The total compatibility score can vary based on the number of criteria satisfied, with a maximum score of 100% if all 25 criteria are met.

Criterions compared:

- Gender
- preferences
- location
- Move in date.
- monthly budget
- Type Of roommate
- Occupancy
- Layout
- Alcohol
- Cleanliness
- Usage of Drugs
- Describe You
- Cooking Habits
- Entertain Friends Frequency
- Roommate Interaction
- Sleeping Hours
- Tobacco
- Work Schedule
- Age

- Career
- Education
- Gender
- Hometown
- Relationship Status
- Interests

4. IMPLEMENTATION

This section thoroughly explores the codebase, examining each module specified during the design process. This section functions as a practical manual, explaining the code's logic, structure, and functionality. The purpose of each code segment is revealed through clear explanations, demonstrating the seamless integration of various modules to achieve preset objectives. This section serves as a comprehensive guide, offering a concrete comprehension of the established system. By providing clear and perceptive explanations, to develop a thorough understanding of how design principles are implemented in the functional codebase, which enhances their understanding of the underlying mechanics involved.

4.1 HOME PAGE

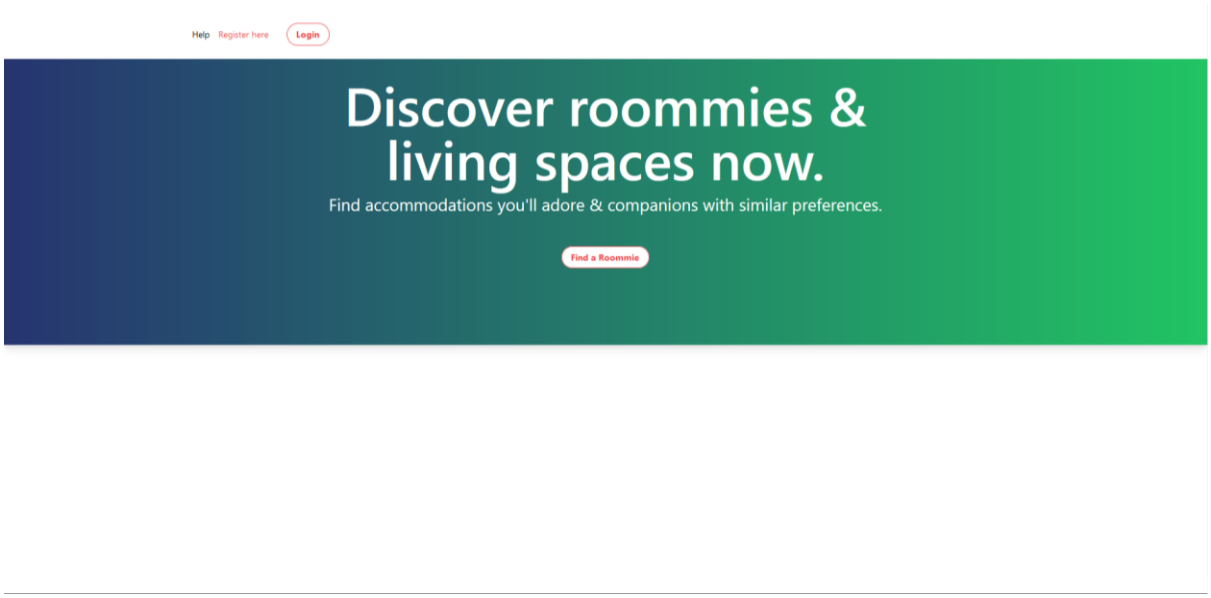


Figure 17: Home Page

The user interface (Figure17) includes a well-placed mobile menu button that is sensitive to user input. This button controls the appearance of a mobile menu, which has navigation links for Help, Register, and Login. In addition, a prominent navigation bar is displayed, showcasing links for Help, Register, and Login. This is achieved by utilizing the Next.js Link component, which enables efficient client-side navigation. The hero part of the homepage features a visually appealing design, highlighted by a gradient backdrop, a widely shown title, a concise subtitle, and a call-to-action button smoothly connected to the sign-in page.

```
<nav className="p-8 bg-white  place-items-center gap-4 ml-16 ">
  <div className="container mx-auto flex justify-between items-center">
    <div className="flex space-x-4">
      <Link href="/helpAndSupportPage" legacyBehavior>
        <span className=" hover:underline visited:text-green-600">
          Help
        </span>
      </Link>
      <Link href="/sign-up">
        <span className="text-red-500 hover:underline visited:text-green-600 mr-4">
          Register here
        </span>
      </Link>
      <Link href="/sign-in">
        <span className="bg-white hover:text-black text-red-500 font-bold py-2 px-4 rounded-full focus:outline-none focus:shadow-outline border-red-500 border-2">
          Login
        </span>
      </Link>
    </div>
  </div>
</nav>

<div className="p-8  bg-gradient-to-r from-[#263470] to-[#21C463] shadow-lg text-center">
  <p className="max-w-6xl mx-auto text-8xl font-semibold mb-4 flex justify-center items-center text-white">
    Discover roommies & living spaces now.
  </p>
  <p className="max-w-6xl mx-auto text-3xl text-white">
    Find accommodations you'll adore & companions with similar
    preferences.
  </p>
  <div className="flex flex-row max-w-4xl mx-auto mt-16 justify-center mb-28">
    <Link href="/sign-in">
```

Figure 18: Home Page Code

Implementation code path:

View:

Roommate-Matching-Platform\front-end\roommate-matching\pages\index.js

4.2 USER REGISTRATION

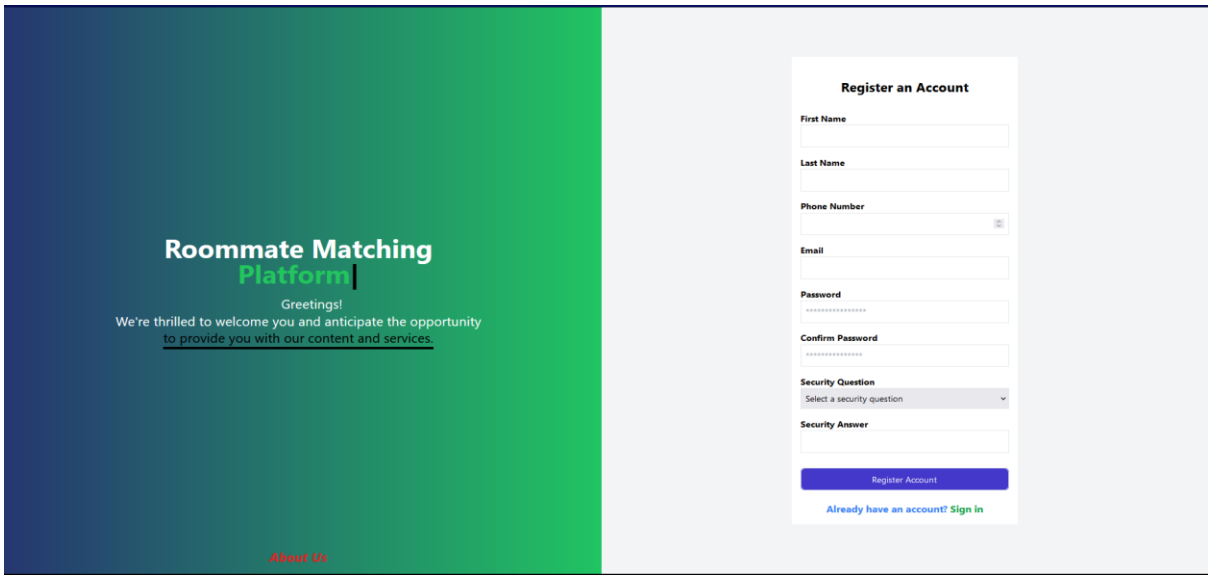


Figure 19: Registration Page

The interface (Figure 19) is structured with a two-column style, showcasing a visually captivating gradient backdrop on the left side. It includes a welcoming message and a hyperlink to get further information about the platform. A registration form is presented on the right side,

with fields for the user's first name, last name, phone number, email, password, and confirmation password. In addition, the user is required to choose a security question from a predetermined selection and submit a response. The form includes real-time validation to verify that the user provides accurate information. Dynamic error messages are presented for each input field in the event of incorrect or missing data. After the form is successfully submitted, the data is transmitted to a local server endpoint for processing.

The code structure incorporates React hooks, including useState and useEffect, to handle state management and side effects. The fetch API is utilized to execute an asynchronous POST request to the server, preserving the user's registration information. (Figure 20)

```
const takevalue = async () => {
  try {
    const response = await fetch("http://localhost:8081/save", {
      method: "POST",
      headers: {
        "Content-Type": "application/json",
        Accept: "application/json",
        "Access-Control-Allow-Origin": "*",
      },
      body: JSON.stringify({
        firstName: formValues.fName,
        lastName: formValues.lName,
        phoneNumber: formValues.phoneNumber,
        email: formValues.email,
        password: formValues.password,
        question,
        answer,
      }),
    });
```

Figure 20: API call for registration

The implemented code incorporates an API call that provides the user data (such as first name, last name, etc.) within the request body as a JSON object. The Java code on the server side gets the JSON object using the *@RequestBody* annotation and performs operations on it. The system does validation, verifies that the security question and answer are not empty, encrypts the password, changes the security question and answer to lowercase, stores the user in the database using the userRepo.save(user) function, and ultimately returns a 200 OK response. (Figure 21)

```java
@CrossOrigin
@PostMapping(value = "/save")
public ResponseEntity<?> saveUser(@RequestBody User user) {
    String message = validation.validateUserDetails(user);
    if(!message.isEmpty()){
        ErrorResponseDto errorResponseDto = new ErrorResponseDto();
        errorResponseDto.setStatusCode(400);
        errorResponseDto.setErrorMessage(message);
        return ResponseEntity.badRequest().body(errorResponseDto);
    }
    if(user.getQuestion().isBlank()){
        message = "Question is null, empty or blank";
    }

    if(user.getAnswer().isBlank()){
        message = "Answer is null, empty or blank";
    }
    String password = user.getPassword();
    user.setPassword(hashPassword(password));
    user.setQuestion(user.getQuestion().toLowerCase());
    user.setAnswer(user.getAnswer().toLowerCase());
    userRepo.save(user);
    return ResponseEntity.ok(user);
}
```

Figure 21: Controller code for registration

To summarize, the React code and the Java code collaborate to transmit user registration details from the frontend to the backend API endpoint. In the backend, the appropriate validations are carried out and the data is processed to be stored in the database.

Implementation code paths:

View:

- *Roommate-Matching-Platform\front-end\roommate-matching\pages\sign-up.js*

Controller:

- *Roommate-Matching-Platform\back-end\rest\src\main\java\com\roommate\app\rest\controller\ApiControllers.java*

Model:

- *Roommate-Matching-Platform\back-end\rest\src\main\java\com\roommate\app\rest\Models\User.java*
- *Roommate-Matching-Platform\back-end\rest\src\main\java\com\roommate\app\rest\Repo\UserRepo.java*

4.3 USER LOGIN

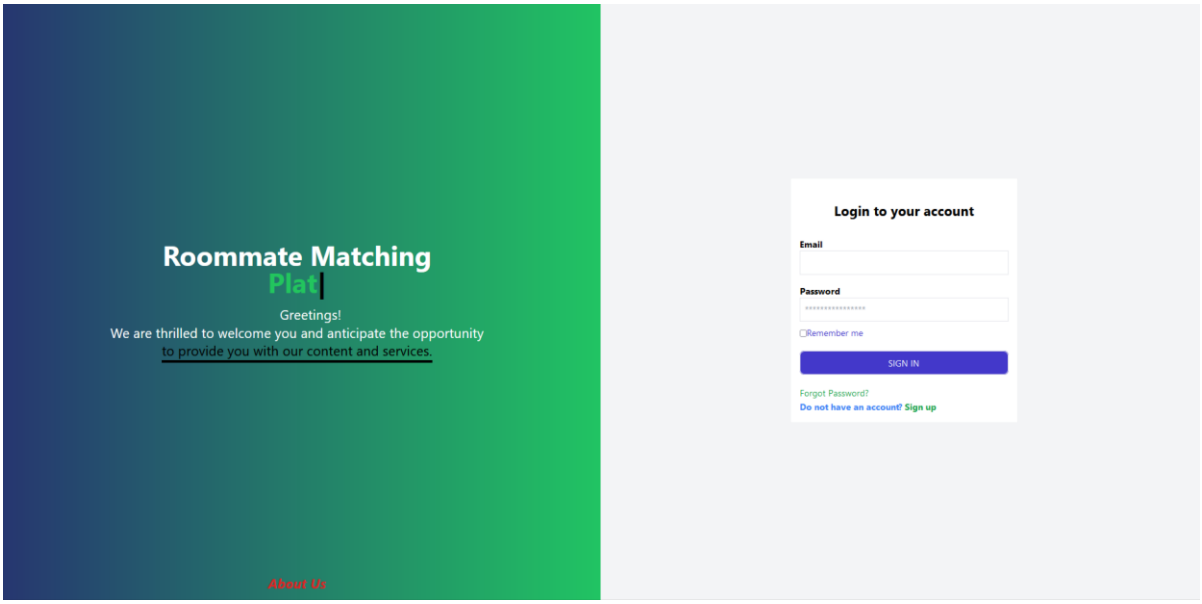


Figure 22: User Login Page

The layout (Figure 22) is divided into two sections: the left section displays a welcoming message, while the right section contains the sign-in form. The form includes input fields for the user's email and password, with real-time validation to provide immediate feedback to users based on their input.

The sign-in process is triggered by the signInUser function, which performs a POST request to the server endpoint "*http://localhost:8081/signIn*" to authenticate the user. The code includes robust error-handling mechanisms, with detailed messages provided to users in case of sign-in failures or network issues. (Figure 23)

```
const apiUrl = "http://localhost:8081/signIn";

const options = {
  method: "POST",
  headers: {
    "Content-Type": "application/json",
    "Access-Control-Allow-Origin": "*",
  },
  body: JSON.stringify(formData),
};
```

Figure 23: API call for Login Process

The frontend's API call establishes a connection with the backends' */signIn* endpoint by transmitting an HTTP POST request. The request contains the user's email and password, which

are inputted by the user on the frontend. Upon getting this request, the backend employs the credentials for validation.

Internally, the backend code verifies the incoming credentials by cross-referencing them with a database. More precisely, it retrieves the person from the database by searching for the email supplied using the *userRepo.findUserByEmail(cred.getEmail())* method. If the user cannot be located or if there is an inconsistency in the password, the backend will respond with a 401 status code, indicating that the authentication has failed. However, if the validation is successful, the backend will give a 200 OK response, along by the user data. (Figure 24)

```
@CrossOrigin
@PostMapping("signIn")
public ResponseEntity<?> signIn(@RequestBody Credentials cred){
    String message = validation.validateCredentials(cred);
    if(!message.isEmpty()){
        ErrorResponseDto errorResponseDto = new ErrorResponseDto();
        errorResponseDto.setStatusCode(400);
        errorResponseDto.setErrorMessage(message);
        return ResponseEntity.badRequest().body(errorResponseDto);
    }
   User user = userRepo.findUserByEmail(cred.getEmail());
   if(user == null){
       return ResponseEntity.status(401).body(user);
   }

   if(user.getEmail().equals(cred.getEmail()) && checkPassword(cred.getPassword(), user.getPassword())){
       return ResponseEntity.ok(user);
   }
   return ResponseEntity.status(401).body(user);
}
```

Figure 24:Login Controller

Upon successful sign-in, the user's ID is stored in a cookie using js-cookie, and the application checks whether the user has completed their profile or preferences. Depending on the completion status, the user is redirected to the appropriate page *(/dashboardPage, /profileSetup, or /preferencePage*).

The code structure ensures a responsive and user-friendly sign-in experience, with attention to detail in terms of user interface design and functionality.

Implementation code paths:

View:

- *Roommate-Matching-Platform\front-end\roommate-matching\pages\sign-in.js*

Controller:

- *Roommate-Matching-Platform\back-end\rest\src\main\java\com\roommate\app\rest\controller\ApiControllers.java*

Model:

- *Roommate-Matching-Platform\back-end\rest\src\main\java\com\roommate\app\rest\Models\User.java*

4.4 PROFILE SETUP

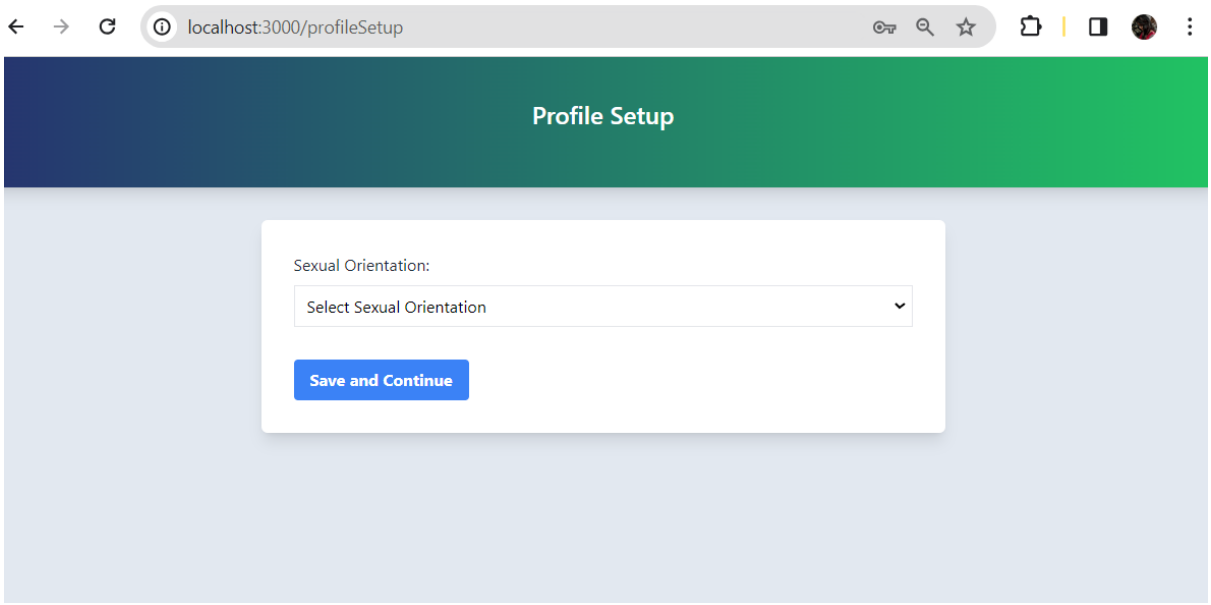


Figure 25: Profile setup UI

The interface (Figure 25) use forms to facilitate the configuration of a user profile. The user interface (UI) is segmented into many parts, including "Sexual Orientation," "Lifestyle Preferences," and "Upload Images." Users may navigate between these sections using buttons labelled "Previous," "Save and Continue," and "Finish Setup." The form has validation logic to verify that mandatory fields are completed prior to advancing to the subsequent section. In addition, the user interface includes a loading page that verifies the user's login status,. When the user clicks the "Finish Setup" button, the component initiates an API call to store the user's profile data. This call is made by sending a POST request to the URL *"http://localhost:8081/saveProfile"* and include the form data in a FormData object. The user interface (UI) utilizes toast alerts to visually indicate if the profile creation procedure was successful or unsuccessful. (Figure 26)

```
const response = await fetch("http://localhost:8081/saveProfile", {
  method: "POST",
  headers: {
    "Access-Control-Allow-Origin": "*",
  },
  body: formDataObject, // Use the FormData object
});
```

Figure 26: API call for saving Profile setup.

```
@PostMapping(value = "/saveProfile")
public ResponseEntity<?> saveProfile(@RequestParam String interests,
                                     @RequestParam long userId,
                                     @RequestParam String sexualOrientation,
                                     @RequestParam MultipartFile profilePicture) throws IOException {
    String message = validation.validateProfile( interests, userId, sexualOrientation, profilePicture);
    if(!message.isEmpty()){
        ErrorResponseDto errorResponseDto = new ErrorResponseDto();
        errorResponseDto.setStatusCode(400);
        errorResponseDto.setErrorMessage(message);
        return ResponseEntity.badRequest().body(errorResponseDto);
    }
    Optional<User> user = userRepo.findById(userId);
    Profile profile  = new Profile();
    profile.setUser(user.orElseThrow(() -> new RuntimeException("User not found")));
    String imageName = saveImage(profilePicture);
    profile.setInterests(interests);
    profile.setSexualOrientation(sexualOrientation);
    profile.setProfilePicture(imageName);
    profileRepo.save(profile);
    return ResponseEntity.ok(profileRepo.findAll());
}
```

Figure 27: Controller method to save Profile setup.

Spring Boot controller function handles the POST request to the "/saveProfile" endpoint. The system accepts form data, which includes fields such as "interests," "userId," "sexualOrientation," and a "profilePicture" as a multipart file. The process starts by verifying the incoming data using a validation object. If the validation is unsuccessful, it will generate a ResponseEntity with a 400 Bad Request status and an accompanying error message.

If the validation is successful, the function proceeds to fetch the user with the designated ID from a database using userRepo. If the user cannot be located, it will raise a RuntimeException. If not, it then generates a new Profile object and links it to the user that was obtained. The procedure saves the profile picture by calling the saveImage method to store the image file, sets the profile information, and saves the profile object database using *profileRepo*. Ultimately, the approach yields a ResponseEntity object with an HTTP status code of 200 OK. (Figure 27)

Implementation code paths:

View:

- *Roommate-Matching-Platform\front-end\roommate-matching\pages\profileSetup.js*

Controller:

- *Roommate-Matching-Platform\back-end\rest\src\main\java\com\roommate\app\rest\controller\ApiControllers.java*

Model:

- *Roommate-Matching-Platform\back-end\rest\src\main\java\com\roommate\app\rest\Models\User.java*
- *Roommate-Matching-Platform\back-end\rest\src\main\java\com\roommate\app\rest\Repo\UserRepo.java*

4.5 LIFESTYLE INFORMATION SETUP

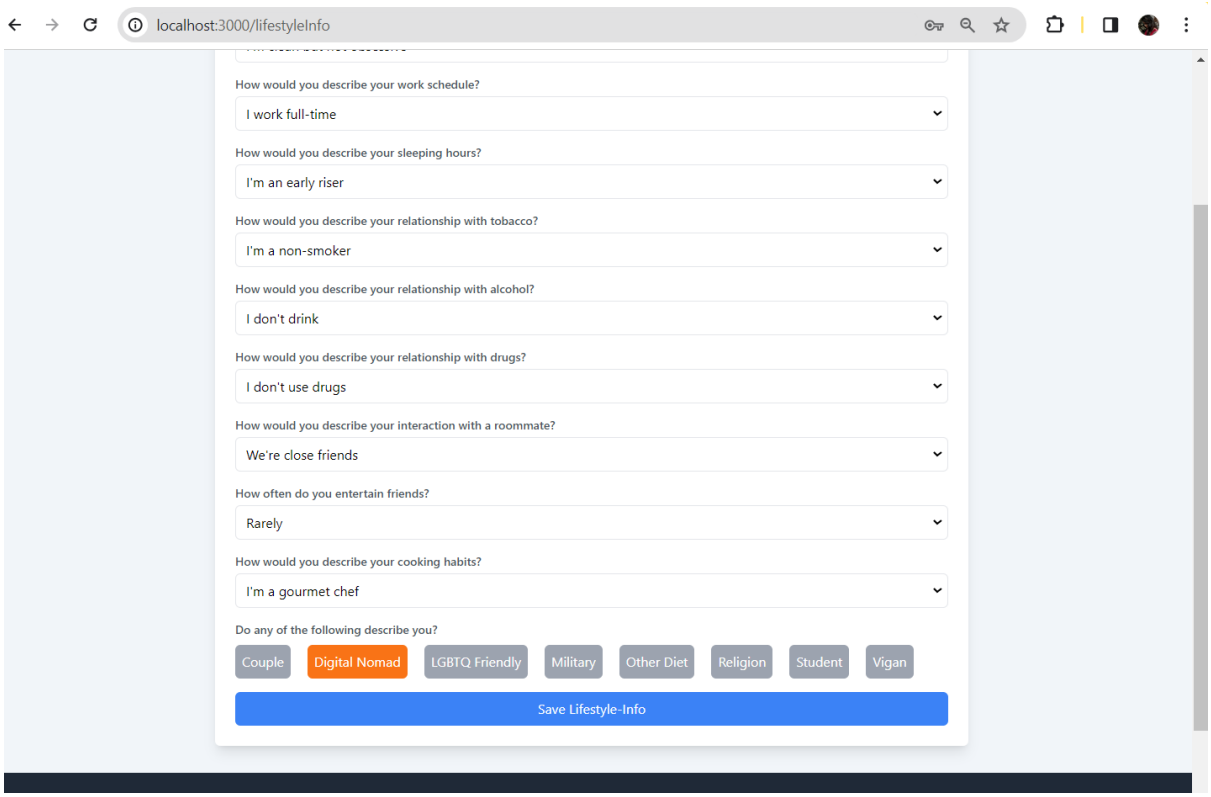


Figure 28: UI for Lifestyle Setup

The user interface (Figure 28) was developed for collecting comprehensive lifestyle data. Users are required to provide their preferences about hygiene, work schedule, sleeping habits, and their position on smoke, alcohol, and drugs. Furthermore, the form contains questions

regarding interactions with roommates, frequency of hosting friends, culinary preferences, and provides other alternatives for users to characterize themselves (such as being a couple, a digital nomad, or LGBTQ-friendly). After the user submits the form, a POST request is sent to the "*http://localhost:8081/saveLifestyleInformation*" endpoint. The component then processes the result and displays either success or error messages.

```
try {
  const response = await fetch(
    "http://localhost:8081/saveLifestyleInformation",
    {
      method: "POST",
      headers: {
        "Content-Type": "application/json",
        "Access-Control-Allow-Origin": "*",
      },
      body: JSON.stringify(formValues),
    }
  );
```

Figure 29: API call to save Lifestyle information.

```
@CrossOrigin
@PostMapping("/saveLifestyleInformation")
public ResponseEntity<?> saveLifestyleInformation(@RequestBody  LifestyleInformationDto lifestyleInformationDto) throws JsonProcessingException
    String message = validation.validateLifestyleInformation(lifestyleInformationDto);
    ErrorResponseDto errorResponseDto = new ErrorResponseDto();
    if(!message.isEmpty()){
        errorResponseDto.setStatusCode(400);
        errorResponseDto.setErrorMessage(message);
        return ResponseEntity.badRequest().body(errorResponseDto);
    }
    Optional<User> user = userRepo.findById(lifestyleInformationDto.getUserId());
    LifestyleInformation lifestyleInformation = new LifestyleInformation();
    lifestyleInformation.setUser(user.orElseThrow());
    lifestyleInformation.setAlcohol(lifestyleInformationDto.getAlcohol());
    lifestyleInformation.setCleanliness(lifestyleInformationDto.getCleanliness());
    lifestyleInformation.setDrugs(lifestyleInformationDto.getDrugs());
    lifestyleInformation.setDescribeYou(lifestyleInformationDto.getDescribeYou());
    lifestyleInformation.setCookingHabits(lifestyleInformationDto.getCookingHabits());
    lifestyleInformation.setEntertainFriendsFrequency(lifestyleInformationDto.getEntertainFriendsFrequency());
    lifestyleInformation.setRoommateInteraction(lifestyleInformationDto.getRoommateInteraction());
    lifestyleInformation.setSleepingHours(lifestyleInformationDto.getSleepingHours());
    lifestyleInformation.setTobacco(lifestyleInformationDto.getTobacco());
    lifestyleInformation.setWorkSchedule(lifestyleInformationDto.getWorkSchedule());
    lifestyleInformationRepo.save(lifestyleInformation);
    return ResponseEntity.ok(lifestyleInformation);
}
```

Figure 30: Contoller method to save Lifestyle information.

The controller method was developed to process HTTP POST requests that are made to the "/*saveLifestyleInformation*" endpoint within a Spring Boot application. This function requires a JSON payload that contains lifestyle information, which is enclosed in a *LifestyleInformationDto* object. The main objective of this strategy is to verify and save the lifestyle data that has been obtained. The validation procedure is triggered by using the

validation.validateLifestyleInformation(lifestyleInformationDto) method, which validates the *LifestyleInformationDto* object. If any validation errors are identified, an error response is created, consisting of a status code of 400 and an error message, and then sent back to the client to indicate a faulty request.(Figure 30)

If the validation is successful, the procedure then proceeds to handle the lifestyle information. The function fetches the corresponding user from the database using the user ID specified in the DTO. Afterwards, a fresh instance of the *LifestyleInformation* entity is generated and filled with information from the DTO, encompassing qualities such as hygiene, work schedule, and social habits.

The *LifestyleInformation* entity is then saved to database using the *lifestyleInformationRepo.save(lifestyleInformation*) method. Ultimately, a *ResponseEntity* is created, which a status code of 200. The *ResponseEntity* is sent back to the client to confirm that the lifestyle information has been stored successfully.

Implementation code paths:

View:

- *Roommate-Matching-Platform\front-end\roommate-matching\pages\lifestyleInfo.js*

Controller:

- *Roommate-Matching-Platform\back-end\rest\src\main\java\com\roommate\app\rest\controller\ApiControllers.java*

Model:

- *Roommate-Matching-Platform\back-end\rest\src\main\java\com\roommate\app\rest\Dto\ LifestyleInformationDto.java*
- *Roommate-Matching-Platform\back-end\rest\src\main\java\com\roommate\app\rest\Models\LifestyleInformation.java*
- *Roommate-Matching-Platform\back-end\rest\src\main\java\com\roommate\app\rest\Repo\LifestyleInformationRepo.java*
- *Roommate-Matching-Platform\back-end\rest\src\main\java\com\roommate\app\rest\Models\User.java*

4.6 PERSONAL INFORMATION SETUP

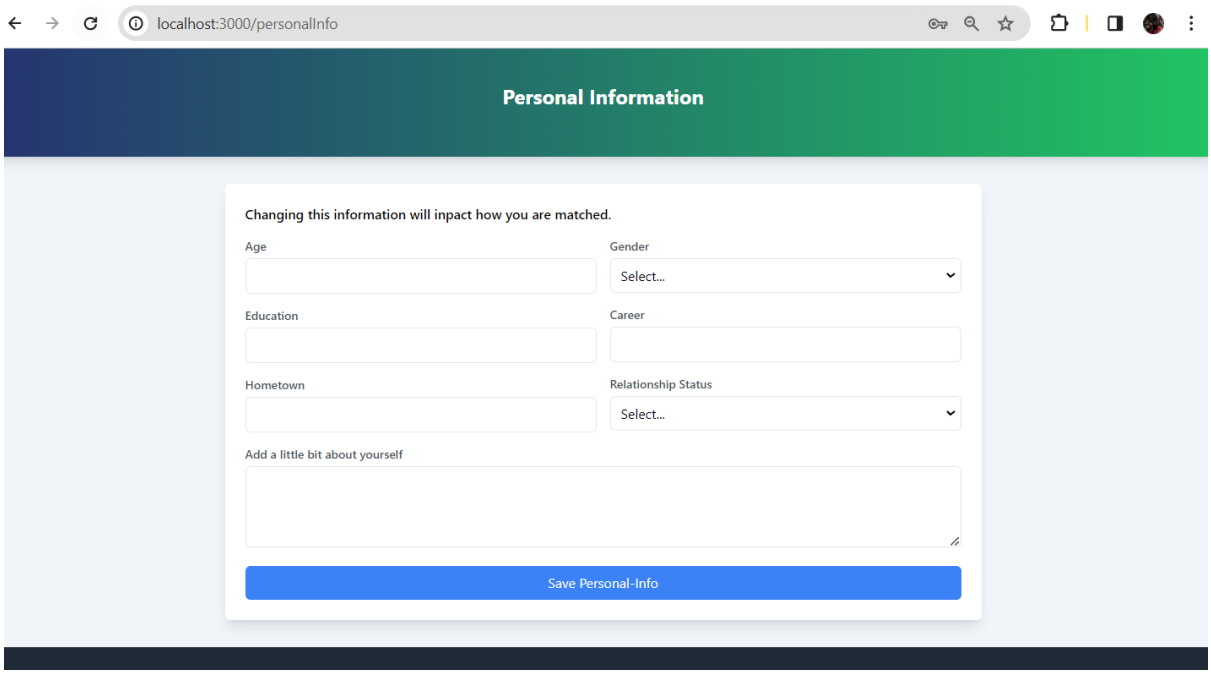


Figure 31: UI for Personal Information Setup

The Personal Information Page allows users to enter personal details, and upon submission, it initiates a POST request to the server located at "http://localhost:8081/savePersonalInformation" in order to persist the data. The component verifies user authentication, upon unsuccessful events is redirected to login page. Page redirects the user to "/lifestyleInfo" upon a successful submission of personal information to database. The user interface has form fields for age, gender, education, career, hometown, and relationship status, accompanied with a self-description.

Implementation code paths:

View:

- *Roommate-Matching-Platform\front-end\roommate-matching\pages\personalInfo.js*

Controller:

- *Roommate-Matching-Platform\back-end\rest\src\main\java\com\roommate\app\rest\controller\ApiControllers.java*

Model:

- *Roommate-Matching-Platform\back-end\rest\src\main\java\com\roommate\app\rest\Models\User.java*
- *Roommate-Matching-Platform\back-end\rest\src\main\java\com\roommate\app\rest\Repo\UserRepo.java*
- *Roommate-Matching-Platform\back-end\rest\src\main\java\com\roommate\app\rest\Models\PersonalInformation.java*
- *Roommate-Matching-Platform\back-end\rest\src\main\java\com\roommate\app\rest\Dto\PersonalInformationDto.java*
- *Roommate-Matching-Platform\back-end\rest\src\main\java\com\roommate\app\rest\Repo\PersonalInformationRepo.java*
- *Roommate-Matching-Platform\back-end\rest\src\main\java\com\roommate\app\rest\Models\LifestyleInformation.java*
- *Roommate-Matching-Platform\back-end\rest\src\main\java\com\roommate\app\rest\Repo\LifestyleInformationRepo.java*
- *Roommate-Matching-Platform\back-end\rest\src\main\java\com\roommate\app\rest\Models\User.java*

4.7 PREFERNCES SETUP

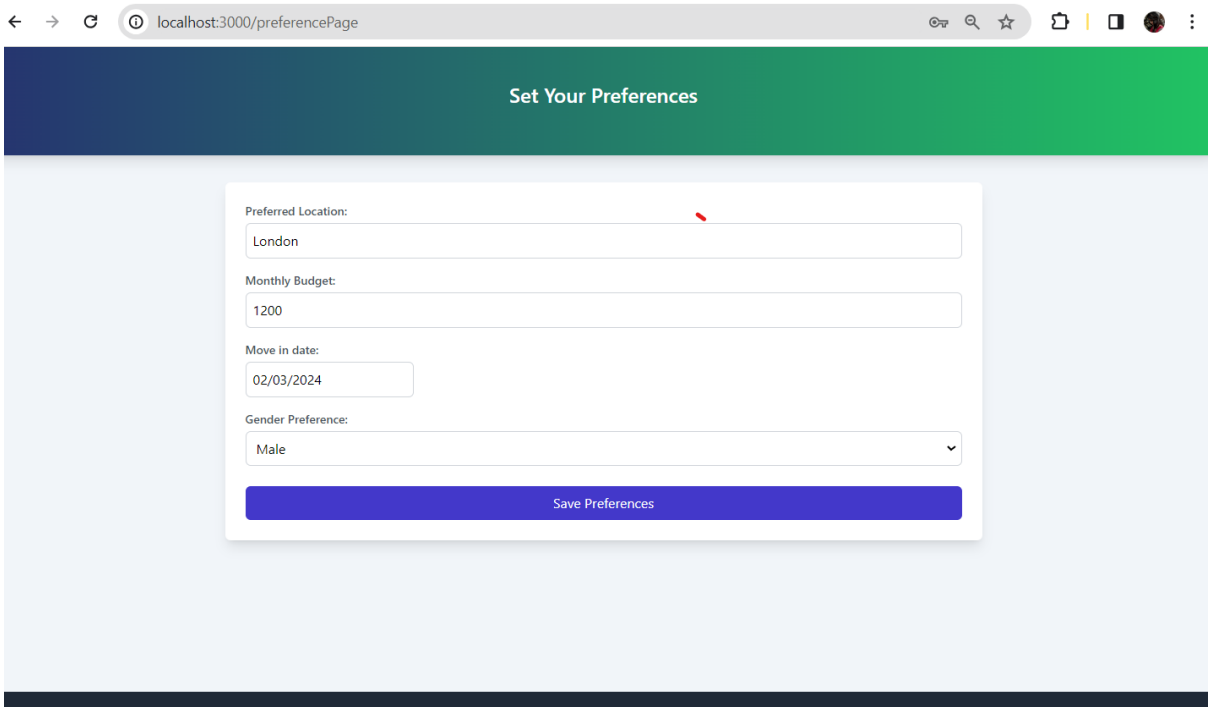


Figure 32: UI for preferences setup

The user interface was developed for configuring user preferences, including desired location, monthly budget, move-in date, and gender preference. The application incorporates form validation logic, and interacts with a backend API to preserve user preferences.

The UI verifies user authentication and redirects to the sign-in page if the user is not authorized. The UI has several essential elements, such as input areas for users to input their desired

location and monthly budget, a date picker for picking the move-in date, and a dropdown menu for specifying gender preference. Clicking the "Save Preferences" button initiates a validation procedure.

```
if (Object.keys(validationErrors).length === 0) {
  try {
    const response = await fetch("http://localhost:8081/savePreference", {
      method: "POST",
      headers: {
        "Content-Type": "application/json",
        Accept: "application/json",
        "Access-Control-Allow-Origin": "*",
      },
      body: JSON.stringify({
        monthlyBudget: preferences.monthlyBudget,
        location: preferences.location,
        moveInDate: preferences.moveInDate,
        genderPreference: preferences.genderPreference,
        userId: jsCookie.get("user"),
      }),
```

Figure 33:API call to save the preferences

Upon successful validation of the form, a POST request is sent to the "*http://localhost:8081/savePreference*" endpoint, including the user's preferences in the request body. Subsequently, the response is managed, exhibiting a notification and navigating the user towards the subsequent page or presenting an error notification in the event of a malfunction.

```
@CrossOrigin
@PostMapping(value = "/savePreference")
public ResponseEntity<?> savePreference(@RequestBody PreferenceDto preferenceDto) {
    String message = validation.validatePreference(preferenceDto);
    if(!message.isEmpty()){
        ErrorResponseDto errorResponseDto = new ErrorResponseDto();
        errorResponseDto.setStatusCode(400);
        errorResponseDto.setErrorMessage(message);
        return ResponseEntity.badRequest().body(errorResponseDto);
    }
    Optional<User> user = userRepo.findById(preferenceDto.getUserId());
    Preference preference = new Preference();
    preference.setGenderPreference(preferenceDto.getGenderPreference());
    preference.setLocation(preferenceDto.getLocation());
    preference.setMoveInDate(preferenceDto.getMoveInDate());
    preference.setMonthlyBudget(preferenceDto.getMonthlyBudget());
    preference.setUser(user.orElseThrow(() -> new RuntimeException("User not found")));


    preferenceRepo.save(preference);
    return ResponseEntity.ok(preferenceRepo.findAll());
}
```

Figure 34: Controller code to save the preferences

The Spring Boot controller method was specifically developed to accept POST requests for saving user preferences. The function accepts a JSON payload that contains user preference data enclosed within a *PreferenceDto* object. The process starts by verifying the submitted preferences through a validation service, examining for any potential problems. In the event of validation failure, a response containing an error message and a status code of 400 (Bad Request) is returned.

After the validation is successfully completed, the procedure proceeds to handle the preference data. This function obtains the user associated with the given user ID from the database. A new Preference entity is instantiated and filled with the preference data, such as gender preference, location, move-in date, and monthly budget. The Preference object is subsequently stored in the database using repository object.

Ultimately, the approach creates a *ResponseEntity* object with a status code of 200 (OK). The *ResponseEntity* is sent back to the client to confirm that the user's preferences have been stored successfully.

Implementation code paths:

View:

- *Roommate-Matching-Platform\front-end\roommate-matching\pages\preferencePage.js*

Controller:

- *Roommate-Matching-Platform\back-end\rest\src\main\java\com\roommate\app\rest\controller\ApiControllers.java*

Model:

- *Roommate-Matching-Platform\back-end\rest\src\main\java\com\roommate\app\rest\Dto\PreferenceDto.java*
- *Roommate-Matching-Platform\back-end\rest\src\main\java\com\roommate\app\rest\Models\Preference.java*
- *Roommate-Matching-Platform\back-end\rest\src\main\java\com\roommate\app\rest\Repo\PreferenceRepo.java*
- *Roommate-Matching-Platform\back-end\rest\src\main\java\com\roommate\app\rest\Models\User.java*

4.8 ROOM SELECTION

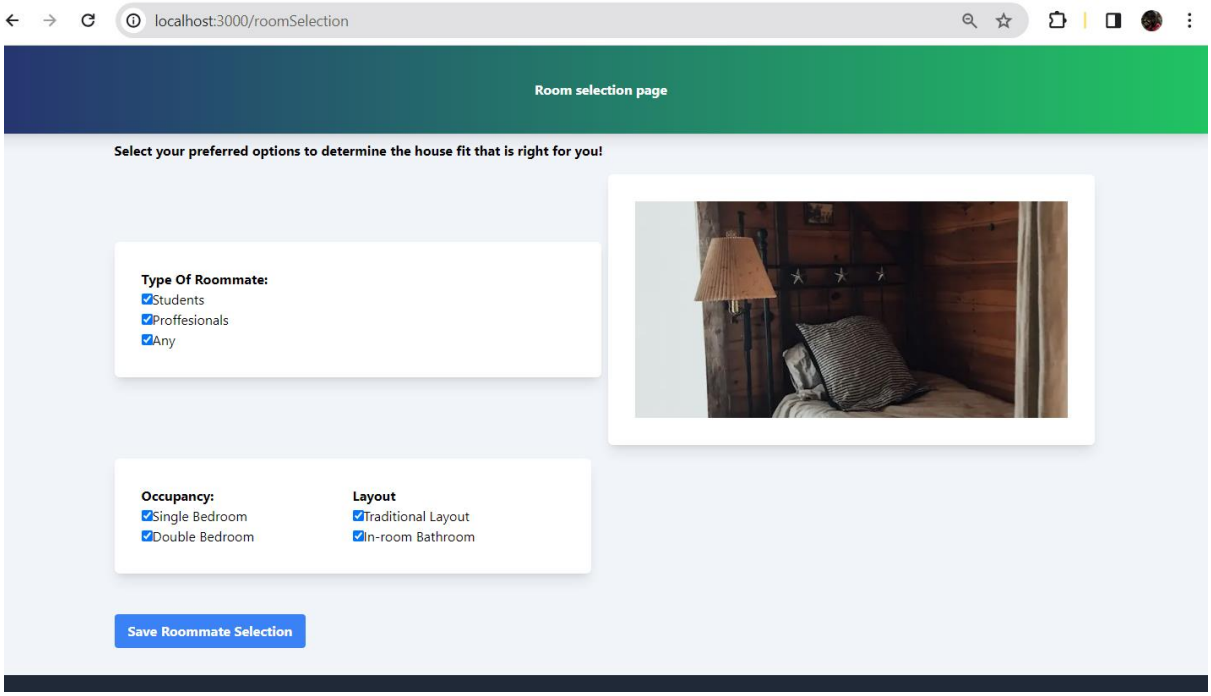


Figure 35: UI for Room selection

The page developed allows users to choose their alternatives to decide the sort of accommodation that best fits their needs. Firstly, the component verifies the user's authentication status and, if not authenticated, leads them to the sign-in page. The code uses state variables to handle the selected choices, form errors, and authentication status. The

component comprises a form including checkboxes that allow the user to pick the roommate type, occupancy (single or double bedroom), and layout preferences.

The chosen preferences are saved in the roommateSelection state, and data validation is conducted before to data submission. The user is shown any form errors, if there are any.

```
if (Object.keys(validationErrors).length === 0) {
  try {
    const userId = jsCookie.get("user") || "";
    const response = await fetch(
      "http://localhost:8081/save-roommate-selection",
      {
        method: "POST",
        headers: {
          "Content-Type": "application/json",
          "Access-Control-Allow-Origin": "*",
        },
        body: JSON.stringify({
          userId,
          typeOfroommate: roommateSelection.typeOfroommate,
          occupancy: roommateSelection.occupancy,
          layout: roommateSelection.layout,
        }),
      }
    );
```

Figure 36: API call to save room selection.

The API call is launched by using the fetch method, which sends a POST request to the specified endpoint "http://localhost:8081/save-roommate-selection". This endpoint is tasked with handling and persisting the given data on the server.

```
@CrossOrigin
@PostMapping("/save-roommate-selection")
public ResponseEntity<?> saveRoommateSelection(@RequestBody RoommateSelectionDto roommateSelectionDto){
    String error = validation.validateRoommateSelection(roommateSelectionDto);
    if(!error.isEmpty()){
        ErrorResponseDto errorResponseDto = new ErrorResponseDto();
        errorResponseDto.setStatusCode(400);
        errorResponseDto.setErrorMessage(error);
        return ResponseEntity.badRequest().body(errorResponseDto);
    }
    Optional<User> user = userRepo.findById(roommateSelectionDto.getUserId());
    RoommateSelection roommateSelection = new RoommateSelection();
    roommateSelection.setTypeOfStudent(roommateSelectionDto.getTypeOfStudent());
    roommateSelection.setLayout(roommateSelectionDto.getLayout());
    roommateSelection.setOccupancy(roommateSelectionDto.getOccupancy());
    roommateSelection.setUser(user.orElseThrow());
    roommateSelectionRepo.save(roommateSelection);
    return ResponseEntity.ok("saved successfully");
}
```

Figure 37: Controller method to save room selection

The spring boot method handles incoming roommate selection data that is received via a POST call to the "/save-roommate-selection" API. The objective is to verify, store, and provide a suitable response to the customer. Upon receiving the *RoommateSelectionDto* object that includes the user's chosen preferences, the function initially conducts validation using a validation component. If any validation issues are detected, an ErrorResponseDto is generated, indicating a status code of 400 (Bad Request) and providing information on the discovered faults. The incorrect answer is subsequently sent back to the client, signifying the requirement for remedial measures to be taken on the given data. If the validation is successful, the procedure then retrieves the user associated with the given *userId*. The *RoommateSelection* entity is subsequently created and filled with the pertinent information extracted from the DTO. The identified user is linked to this entity, which is then stored in the database using the *roommateSelectionRepo.save(roommateSelection)* method. Ultimately, the approach creates a ResponseEntity object with a status code of 200 (OK). The *ResponseEntity* is sent back to the client to confirm that the user's room selection have been stored successfully. The server's response is subsequently evaluated for success, with a favorable outcome denoted by a status code of 200. Under such circumstances, after the user has successfully saved their preferences, users are automatically transferred to the dashboard page.

Implementation code paths:

View:

- *Roommate-Matching-Platform\front-end\roommate-matching\pages\roomSelection.js*

Controller:

- *Roommate-Matching-Platform\back-end\rest\src\main\java\com\roommate\app\rest\controller\OtherApiControllers.java*

Model:

- *Roommate-Matching-Platform\back-end\rest\src\main\java\com\roommate\app\rest\Dto\RoommateSelectionDto.java*
- *Roommate-Matching-Platform\back-end\rest\src\main\java\com\roommate\app\rest\Models\RoommateSelection.java*
- *Roommate-Matching-Platform\back-end\rest\src\main\java\com\roommate\app\rest\Repo\RoommateSelectionRepo.java*
- *Roommate-Matching-Platform\back-end\rest\src\main\java\com\roommate\app\rest\Models\User.java*

4.9 DASHBOARD

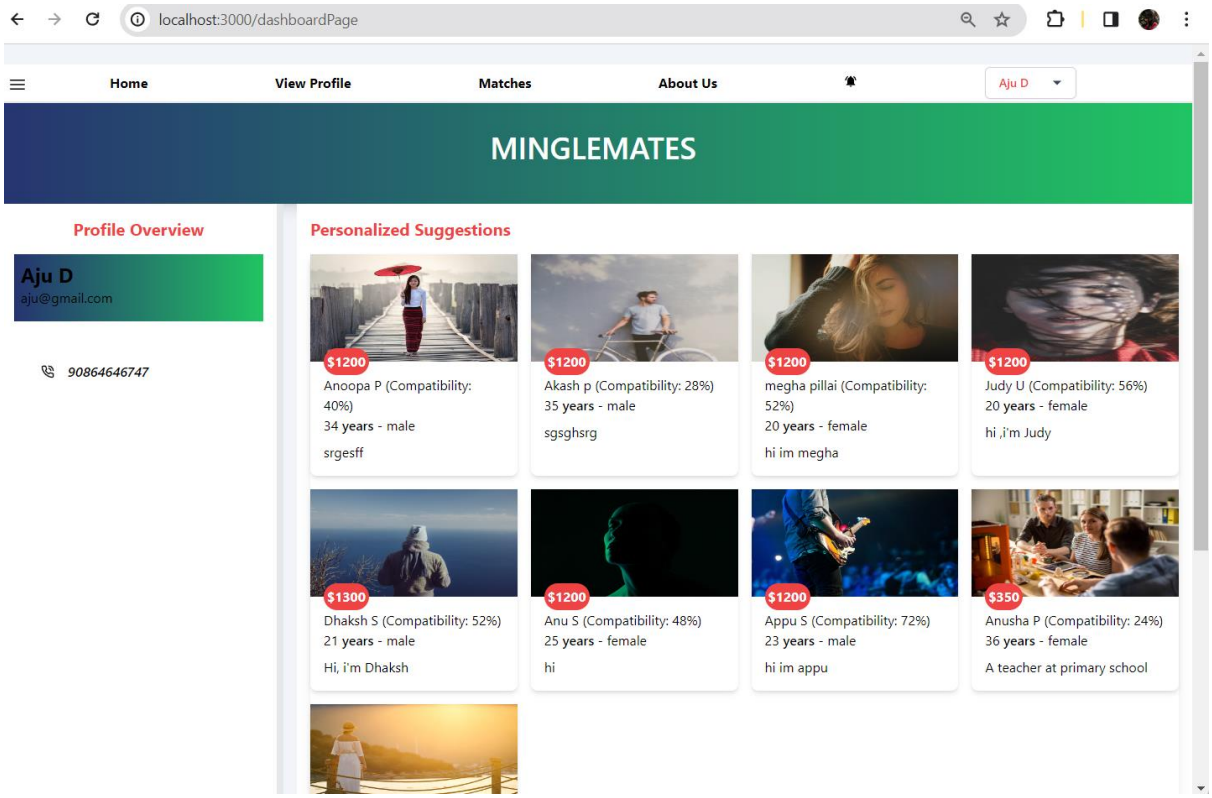


Figure 38:Dashboard page

The dashboard serves as the primary page for the Minglemate application. The application has components such as a navigation bar, a user profile overview, tailored roommate recommendations, and a sidebar with supplementary navigation choices.

Below is a summary of the main features provided by this page:

Navigation Bar: The element consists of a navigation bar that contains hyperlinks to several aspects of the programme, including the dashboard, user profile, matches, and about us. The website is designed to be flexible, meaning it can adapt to different screen sizes. It includes a mobile menu that can be easily switched on and off.

User Authentication: The component verifies the user's authentication status upon initialization. If the user is not authorized, they are sent to the sign-in page. This guarantees that only verified users may have access to the material.

Profile Overview: The user's profile summary is presented on the left-hand side of the page. The information comprises the user's name, email address, phone number, The profile details are retrieved via the backend API.

Roommate Recommendations: Customized roommate recommendations are presented on the right-hand side of the page. The component utilizes API calls to retrieve compatibility data and provide roommate suggestions according to the user's choices. The information on the roommate's name, age, gender, budget, and compatibility % is displayed. By clicking on a roommate's profile image or name, the user is sent to the relevant profile page.

Sidebar: The component features a sidebar that can be switched on or off on smaller displays. The application comprises hyperlinks to several parts, including matches, messages, comments, help and support, testimonial submission, and log out. The sidebar improves the user's ability to navigate.

Logout Functionality: The component offers a logout button that eliminates the user's cookie and sends them to the sign-in page.

API Calls: The component initiates many API calls to retrieve data from the backend. These requests include fetching roommate suggestions, user profile data, and the most recent messages.

Dropdown for Notifications: Notifications for new messages are shown in a dropdown menu. The component retrieves the most recent messages and exhibits them.

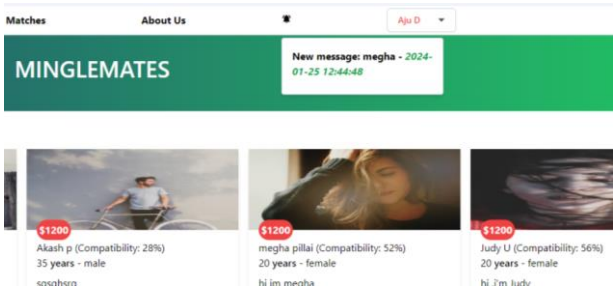


Figure 39: Notifications

Below is an overview of the API calls made in the dashboard page:

Retrieve recommendations for potential roommates:

- Objective: Obtains customized roommate recommendations by considering the user's preferences.
- URL: http://localhost:8081/api/roommates/matches/${userId}

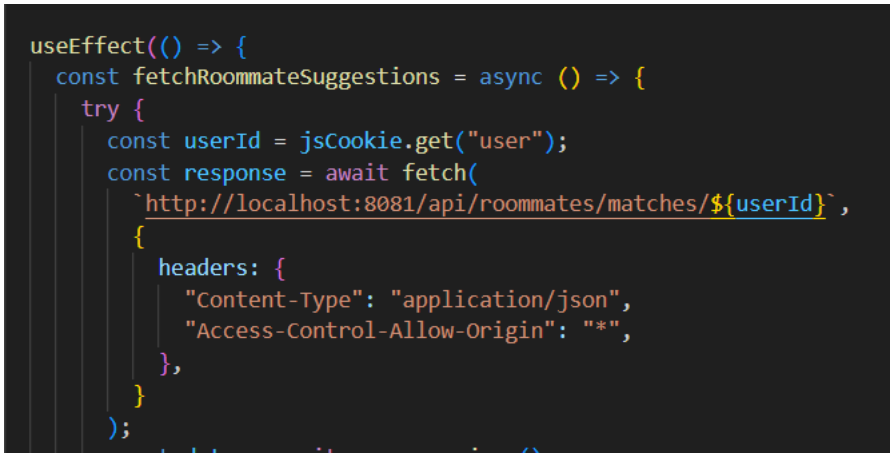


Figure 40: API call to get matching roommates

- The retrieved data is saved in the *roommateSuggestions* state. Furthermore, profile photographs for each suggested roommate are retrieved.

Retrieve user profile information:

- Objective: Retrieves the user's profile data, encompassing fundamental details and profile image.
- URL: http://localhost:8081/profile/${userId}

Figure 41: API call to get profile data of the current user

Retrieve Compatibility Data:

- Objective: Retrieves compatibility info for the user.
- URL: http://localhost:8081/api/roommates/matches-compatibility/${userId}

```
useEffect(() => {
  const fetchCompatibilityData = async () => {
    try {
      const response = await fetch(
        `http://localhost:8081/api/roommates/matches-compatibility/${userId}`,
        {
          headers: {
            "Content-Type": "application/json",
            "Access-Control-Allow-Origin": "*",
          },
        }
      );
      const data = await response.json();
      setCompatibilityData(data);
    } catch (error) {
      console.error("Error fetching compatibility data:", error);
    }
  };
```

Figure 42: API call to get the compatibility with potential roommates.

- The compatibility data is saved in the *compatibilityData* state.

Retrieve the most recent messages:

- Objective: Retrieves the most recent messages for the user.
- URL: http://localhost:8081/getLatestMessage/${userId}

```
useEffect(() => {
  const fetchLatestMessages = async () => {
    try {
      const userId = jsCookie.get("user");
      const response = await fetch(
        `http://localhost:8081/getLatestMessage/${userId}`,
        {
          headers: {
            "Content-Type": "application/json",
            "Access-Control-Allow-Origin": "*",
          },
        }
      );

      if (response.ok) {
        const data = await response.json();
        setLastMessages(data || []);
      } else {
        console.error("Failed to fetch latest messages");
      }
    } catch (error) {
      console.error("Error fetching latest messages:", error);
    }
  };
```

Figure 43: API call to get the latest message

- The received messages are saved in the lastMessages state.

These API requests are essential for retrieving data dynamically, such as customized roommate recommendations, user profile data, compatibility specifics, and the most recent communications. The data acquired from these calls is utilized to refresh the state of the component, facilitating the display of current information on the user's dashboard.

```
@CrossOrigin
@GetMapping("/matches/{userId}")
public ResponseEntity<?> findRoommateMatches(@PathVariable long userId) {
    String error = validation.validateUserId(userId);
    if(!error.isEmpty()){
        ErrorResponseDto errorResponseDto = new ErrorResponseDto();
        errorResponseDto.setStatusCode(400);
        errorResponseDto.setErrorMessage(error);
        return ResponseEntity.badRequest().body(errorResponseDto);
    }
    return ResponseEntity.ok().body(roommateMatcherService.findRoommates(userId));
}
```

Figure 44: Controller method to get roommates for matching

The *findRoommateMatches* controller method in the Spring Boot application, find out suitable roommates mates for a user, as indicated by the given userId. The process starts by verifying the input userId to ensure it satisfies specific requirements. If the validation is unsuccessful, it generates and provides a poor request response that includes information about the validation problem.

Upon successful input validation, the method invokes the *findRoommates* method of the *roommateMatcherService* to get and compile data on compatible roommates for the given user. The outcome is subsequently enclosed within a *ResponseEntity* entity, and an HTTP 200 (OK) status is assigned to indicate a successful response. The response, which includes the data for matching roommates, is sent back to the client.

```
@CrossOrigin
@GetMapping("/matches-compatibility/{userId}")
public ResponseEntity<?> findRoommateMatchesCompatibility(@PathVariable long userId){
    String error = validation.validateUserId(userId);
    if(!error.isEmpty()){
        ErrorResponseDto errorResponseDto = new ErrorResponseDto();
        errorResponseDto.setStatusCode(400);
        errorResponseDto.setErrorMessage(error);
        return ResponseEntity.badRequest().body(errorResponseDto);
    }
    return ResponseEntity.ok().body(roommateMatcherService.compatibility(
            roommateMatcherService.findRoommates(userId), userId));
}
```

Figure 45: Controller method to get the roommate compatibility

The *findRoommateMatchesCompatibility* method was developed exclusively to process GET requests at the *"/matches-compatibility/{userId}"* endpoint. The main objective of this technique is to identify roommate matches for a user based on their specified userId, using compatibility ratings. The method uses the *findRoommates* method of the *roommateMatcherService* in order to obtain information about prospective roommates for the given user. Afterwards, it calls the compatibility method from the same service, providing the list of roommates and the user ID as parameters. This approach computes compatibility ratings for each roommate based on compatibility algorithm developed.

The results, which includes the roommate matches and their compatibility ratings, is stored within a ResponseEntity object, and an HTTP 200 (OK) status is assigned to indicate a successful response. The answer, which includes the compatibility data, is subsequently sent back to the client.

```
@CrossOrigin
@GetMapping("/getLatestMessage/{userId}")
public ResponseEntity<?> getLatestMessage(@PathVariable Long userId){
    return messageService.getMessage(userId);
}
```

Figure 46:Controller code to get the latest mesaage

The "*getLatestMessage*" function retrieves the most recent messages for a specific user ID.

Implementation code paths:

View:

- *Roommate-Matching-Platform\front-end\roommate-matching\pages\dashboardPage.js*

Controller:

- *Roommate-Matching-Platform\back-end\rest\src\main\java\com\roommate\app\rest\controller\RoommateContoller.java*
- *Roommate-Matching-Platform\back-end\rest\src\main\java\com\roommate\app\rest\controller\MessagingController.java*

Model:

- *Roommate-Matching-Platform\back-end\rest\src\main\java\com\roommate\app\rest\Dto\RoommateSelectionDto.java*
- *Roommate-Matching-Platform\back-end\rest\src\main\java\com\roommate\app\rest\Dto\MessageDto.java*
- *Roommate-Matching-Platform\back-end\rest\src\main\java\com\roommate\app\rest\Models\RoommateSelection.java*
- *Roommate-Matching-Platform\back-end\rest\src\main\java\com\roommate\app\rest\Models\Message.java*
- *Roommate-Matching-Platform\back-end\rest\src\main\java\com\roommate\app\rest\Repo\RoommateSelectionRepo.java*
- *Roommate-Matching-Platform\back-end\rest\src\main\java\com\roommate\app\rest\Repo\MessageRepo.java*
- *Roommate-Matching-Platform\back-end\rest\src\main\java\com\roommate\app\rest\Models\User.java*

4.10 PROFILE PAGE and UPDATING PROFILE DETAILS

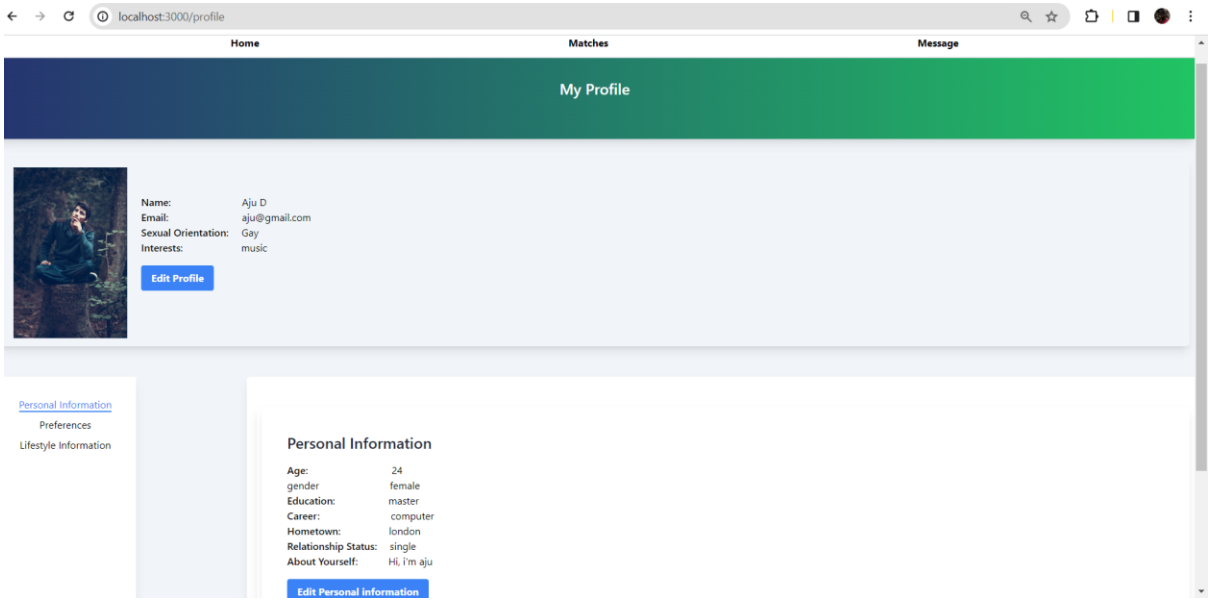


Figure 47: Profile page UI of current user

The page displays user profile data. The application utilizes state management through the usage of React hooks. Upon initialization, the component retrieves user data and photos from a server, while also verifying user authentication. The website displays a responsive design that includes a mobile menu toggle and showcases personal, preference, and lifestyle information (Can also edit those information). Users have the ability to modify specific information by utilizing navigation buttons, which provide a full user profile experience.

The *getProfile* function in controller endpoint specifically created to handle a GET request for fetching a user's profile information using their id. The procedure initially verifies the incoming id input by employing a validation object. If the validation is unsuccessful, the appropriate status code and error message is send as response. Subsequently, method fetch a user from the database using *userRepo* instance by using the given id. Afterwards, the procedure searches for a matching Profile entity in the *profileRepo*. If a profile cannot be located, it generates an additional error response indicating the absence of the user's profile. Upon successful retrieval of the user and profile, the technique proceeds to acquire supplementary details, including preferences, lifestyle information, and personal information linked to the user. The retrieved data is subsequently added into a *ProfileDto* entity, Ultimately, the method yields a *ResponseEntity* containing the created *ProfileDto* entity, indicating a successful response that includes the user's profile details. These details are used to display the profile page as shown in figure.

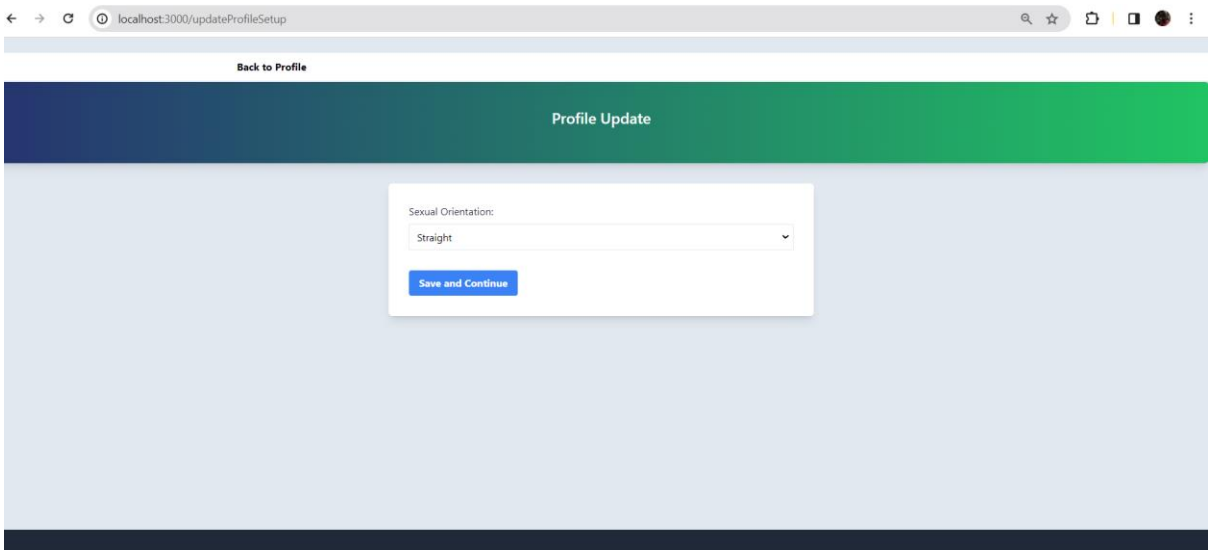


Figure 48: Updating profile information

Figure shows the page developed to update user profile data. The application updates user data, including sexual orientation, lifestyle preferences, and profile images. The user traverses these areas, and form validation guarantees the integrity of the data.

```
const response = await fetch("http://localhost:8081/updateProfile", {
  method: "PUT",
  headers: {
    "Access-Control-Allow-Origin": "*",
  },
  body: formDataObject, // Use the FormData object
});
```

Figure 49:API call to update the profile

After finishing, the user can send the revised profile, which will initiate a PUT request to the server (Figure).

```
@CrossOrigin
@PutMapping(value = "/updateProfile")
public ResponseEntity<?> updateProfile(@RequestParam String interests,
                                       @RequestParam long userId,
                                       @RequestParam String sexualOrientation,
                                       @RequestParam MultipartFile profilePicture) throws IOException {
    Profile profile  = profileRepo.findProfileByUserId(userId);
    String imageName = "";
    if(profilePicture != null){
        imageName = saveImage(profilePicture);
    }
    if(!interests.isBlank()){
        profile.setInterests(interests);
    }
    if(!sexualOrientation.isBlank()){
        profile.setSexualOrientation(sexualOrientation);
    }
    if(!imageName.isBlank()){
        profile.setProfilePicture(imageName);
    }

    profileRepo.save(profile);
    return ResponseEntity.ok(profileRepo.findAll());
}
```

Figure 50: Controller method to update the profile

The *updateProfile* method is responsible for managing the modification of user profiles. The function accepts arguments such as interests, userId, sexualOrientation, and profilePicture (a file). The process involves retrieving the current profile, modifying the relevant values using non-empty parameters, storing a new profile image if one is available, and providing a response with the modified profiles. The figure shows the Sexual Orientation of the user is updated.

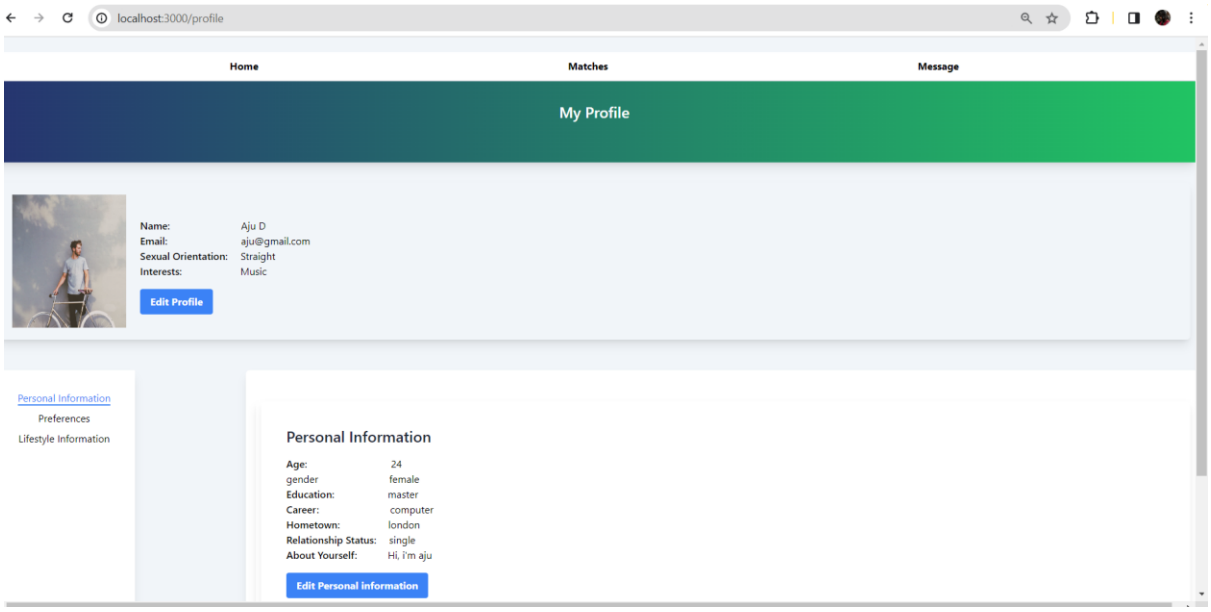


Figure 51: User profile got updated

Implementation code paths:

View:

- *Roommate-Matching-Platform\front-end\roommate-matching\pages\profile.js*
- *Roommate-Matching-Platform\front-end\roommate-matching\pages\updateProfileSetup.js*

Controller:

- *Roommate-Matching-Platform\back-end\rest\src\main\java\com\roommate\app\rest\controller\ApiControllers.java*

Model:

- *Roommate-Matching-Platform\back-end\rest\src\main\java\com\roommate\app\rest\Models\User.java*
- *Roommate-Matching-Platform\back-end\rest\src\main\java\com\roommate\app\rest\Repo\UserRepo.java*
- *Roommate-Matching-Platform\back-end\rest\src\main\java\com\roommate\app\rest\Models\PersonalInformation.java*
- *Roommate-Matching-Platform\back-end\rest\src\main\java\com\roommate\app\rest\Dto\PersonalInformationDto.java*
- *Roommate-Matching-Platform\back-end\rest\src\main\java\com\roommate\app\rest\Repo\PersonalInformationRepo.java*

4.11 UPDATING PREFERENCES, PERSONAL INFORMATION and LIFESTYLE INFORMATION

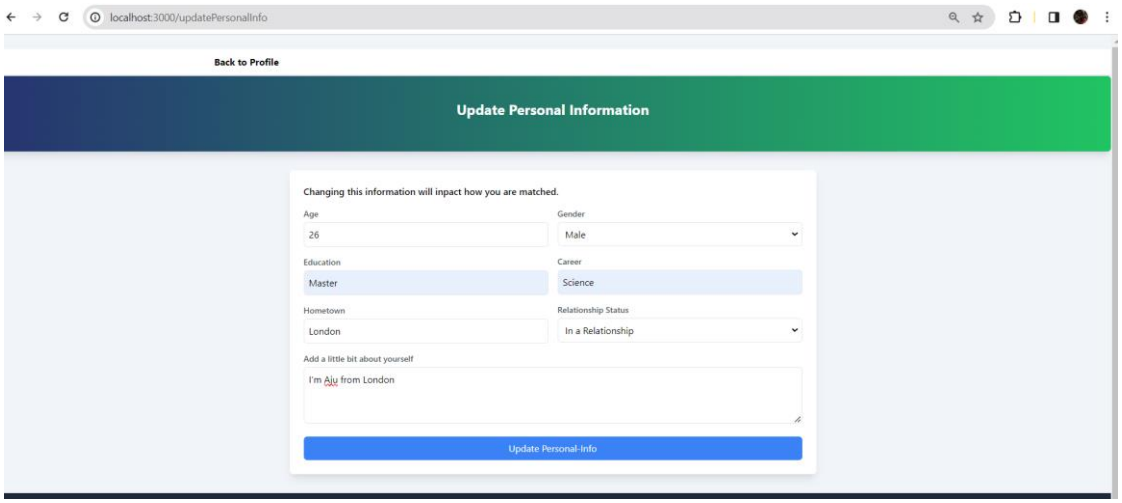


Figure 52: Updating Personal Information

The page utilizes state management to track the authentication status of the user and regulate the appearance of a mobile menu. the page form enables users to edit a range of personal facts, including age, gender, education, job, hometown, relationship status, and a personalized description. A crucial aspect of this page is the incorporation of an API request to modify the user's data on the server.

```
try {
  const response = await fetch(
    "http://localhost:8081/updatePersonalInformation",
    {
      method: "PUT",
      headers: {
        "Content-Type": "application/json",
      },
      body: JSON.stringify(formData),
    }
  );
```

Figure 53: API call to update the personal information.

Upon form submission, the user triggers a PUT request to the backend server located at "*http://localhost:8081/updatePersonalInformation*" The request body contains the updated

data. This API call guarantees the precise synchronisation of the user's information with the server's database.

```
@CrossOrigin
@PutMapping("/updatePersonalInformation")
public ResponseEntity<?> updatePersonalInformation(@RequestBody PersonalInformationDto personalInfo
    PersonalInformation personalInformation = personalInformationRepo.
            findPersonalInformationByUserId(personalInformationDto.getUserId());
    if(personalInformationDto.getAge() > 0){
        personalInformation.setAge(personalInformationDto.getAge());
    }
    if(!personalInformationDto.getCareer().isBlank()){
        personalInformation.setCareer(personalInformationDto.getCareer());
    }
    if(!personalInformationDto.getGender().isBlank()){
        personalInformation.setGender(personalInformationDto.getGender());
    }
    if(!personalInformation.getAboutYourself().isBlank()){
        personalInformation.setAboutYourself(personalInformation.getAboutYourself());
    }
    if(!personalInformationDto.getEducation().isBlank()){
        personalInformation.setEducation(personalInformationDto.getEducation());
    }
    if(!personalInformationDto.getRelationshipStatus().isBlank()){
        personalInformation.setRelationshipStatus(personalInformationDto.getRelationshipStatus());
    }
    if(!personalInformationDto.getHometown().isBlank()){
        personalInformation.setHometown(personalInformationDto.getHometown());
    }
    personalInformationRepo.save(personalInformation);
    return ResponseEntity.ok(personalInformation);
```

Figure 54: Controller method to update the personal Information

The *updatePersonalInformation* endpoint in controller is developed to handle PUT requests for updating a user's personal information. The procedure retrieves the user ID and personal information data from the given *PersonalInformationDto*. Subsequently, it fetches the current personal data linked to the user ID and proceeds to modify certain fields if valid non-null values are supplied. After the changes, the altered personal information is stored again in the database. The endpoint delivers a *ResponseEntity* that includes the modified personal information, confirming the successful completion of the update operation. The Figure shows the Personal Information of the user updated.

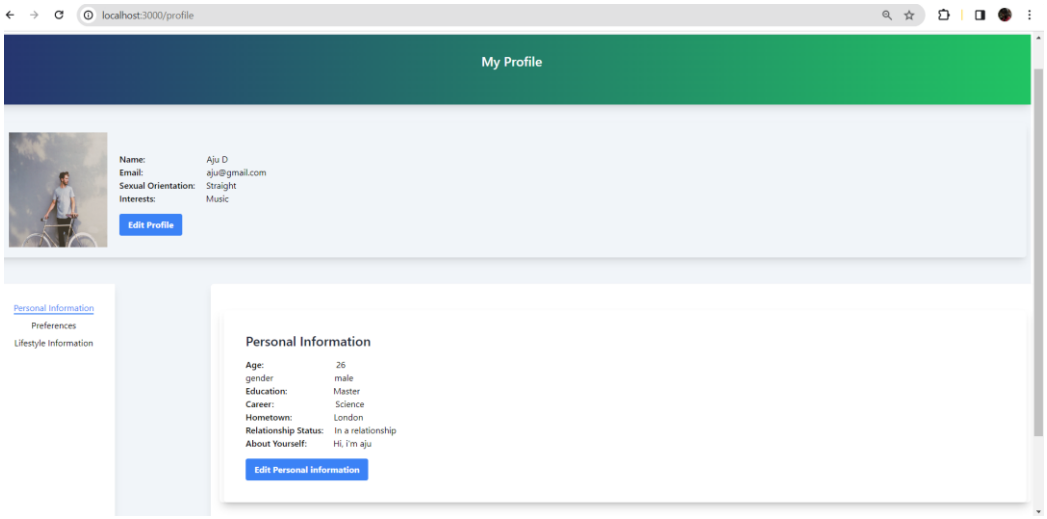


Figure 55: Personal Information Updated

Likewise, modifications were implemented to preferences and lifestyle information. The provided screenshots depict the alterations made in the user interface.

Updating Preferences

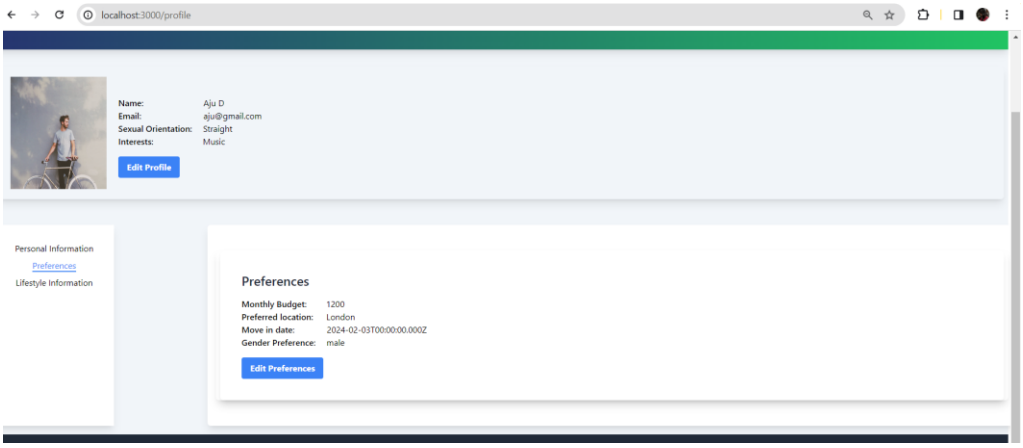


Figure 56: Preferences before updating

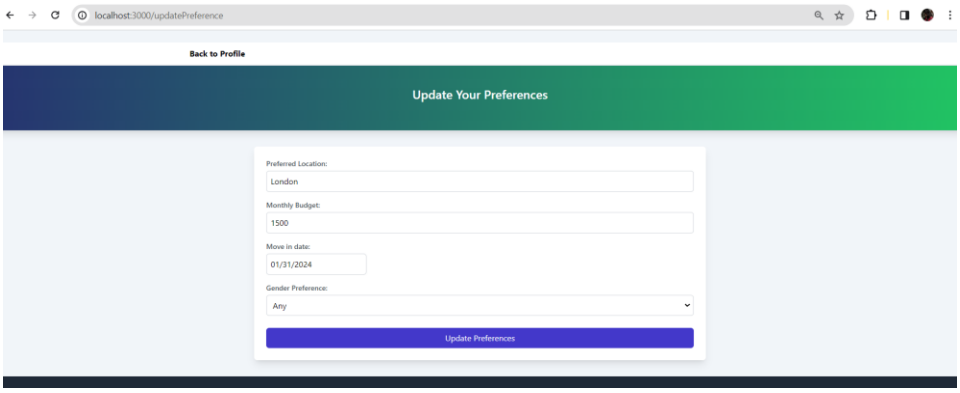


Figure 57: Updating Preferences

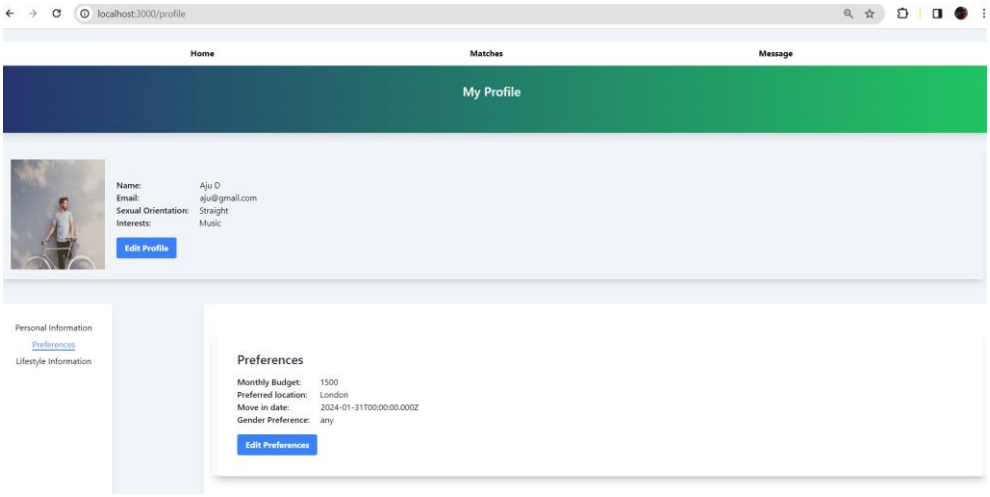


Figure 58: Preferences updated

Updating Lifestyle Information

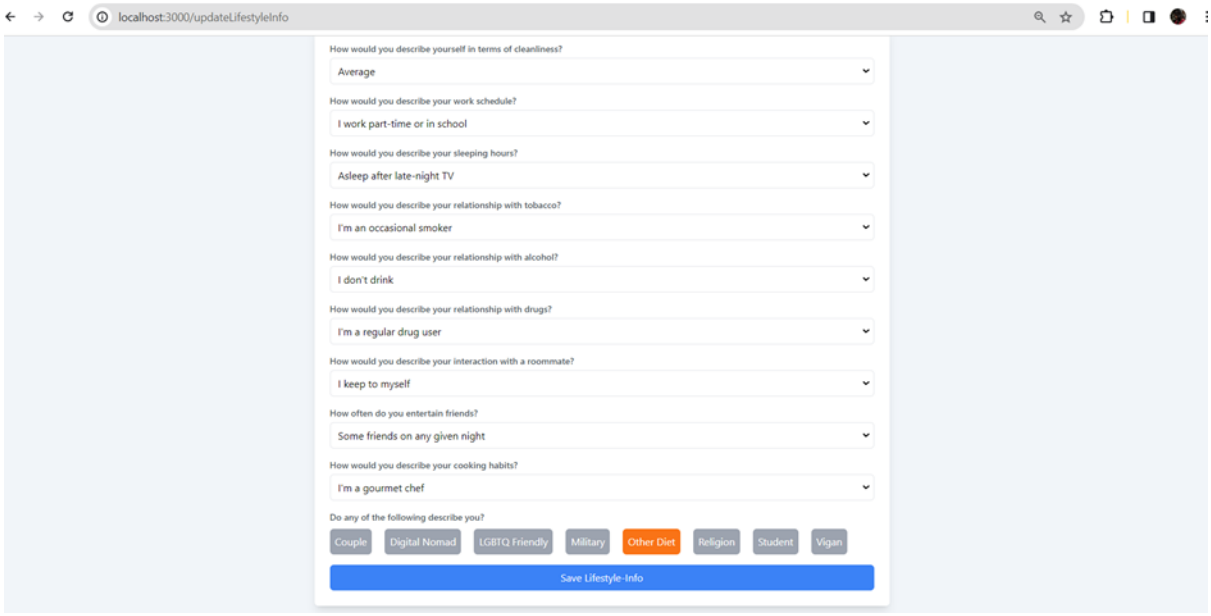


Figure 59: Updating Lifestyle Information

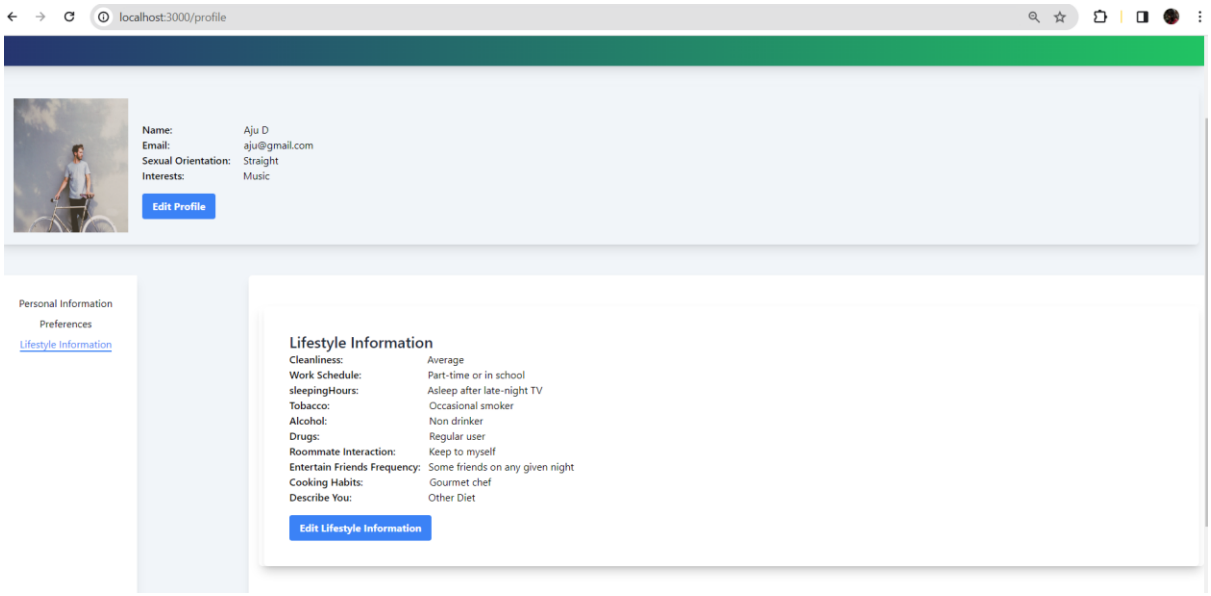


Figure 60: Lifestyle Information Updated

Implementation code paths:

View:

- *Roommate-Matching-Platform\front-end\roommate-matching\pages\profile.js*
- *Roommate-Matching-Platform\front-end\roommate-matching\pages\updateLifestyleInfo.js*
- *Roommate-Matching-Platform\front-end\roommate-matching\pages\updatePersonalInfo.js*
- *Roommate-Matching-Platform\front-end\roommate-matching\pages\updatePreference.js*

Controller:

- *Roommate-Matching-Platform\back-end\rest\src\main\java\com\roommate\app\rest\controller\ApiControllers.java*

Model:

- *Roommate-Matching-Platform\back-end\rest\src\main\java\com\roommate\app\rest\Models\User.java*
- *Roommate-Matching-Platform\back-end\rest\src\main\java\com\roommate\app\rest\Repo\UserRepo.java*
- *Roommate-Matching-Platform\back-end\rest\src\main\java\com\roommate\app\rest\Dto\LifestyleInformationDto.java*
- *Roommate-Matching-Platform\back-end\rest\src\main\java\com\roommate\app\rest\Dto\PersonalInformationDto.java*
- *Roommate-Matching-Platform\back-end\rest\src\main\java\com\roommate\app\rest\Dto\PreferenceDto.java*
- *Roommate-Matching-Platform\back-end\rest\src\main\java\com\roommate\app\rest\Models\LifestyleInformation.java*

- *Roommate-Matching-Platform\back-end\rest\src\main\java\com\roommate\app\rest\Models\PersonalInformation.java*
- *Roommate-Matching-Platform\back-end\rest\src\main\java\com\roommate\app\rest\Models\Preference.java*
- *Roommate-Matching-Platform\back-end\rest\src\main\java\com\roommate\app\rest\Repo\LifestyleInformationRepo.java*
- *Roommate-Matching-Platform\back-end\rest\src\main\java\com\roommate\app\rest\Repo\PersonalInformationRepo.java*
- *Roommate-Matching-Platform\back-end\rest\src\main\java\com\roommate\app\rest\Repo\PreferenceRepo.java*

4.12 MESSAGING

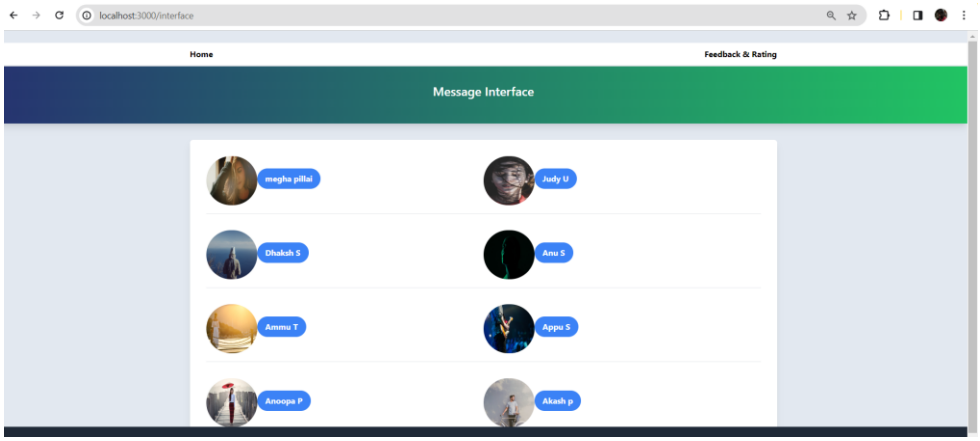


Figure 61: Messaging Interface UI

This page includes a dynamic grid of roommate recommendations, with each recommendation showcasing a profile photograph and a button to establish a connection with the suggested roommate.

```
const data = await response.json();
setRoommateSuggestions(data); // Assuming data is an array of suggestions
setLoading(false);
jsCookie.set("roommateId", data.id);
const imagePromises = data.map(async (roommate) => {
  const imagesResponse = await fetch(
    `http://localhost:8081/profile-picture/${roommate.id}`
  );
  if (imagesResponse.ok) {
    const imagesData = await imagesResponse.blob();
    setUserImages((prevImages) => ({
      ...prevImages,
      [roommate.id]: URL.createObjectURL(imagesData),
```

Figure 62: API call to get Profile picture of rommates

The retrieval of these information is accomplished by making API requests to a backend service. On selecting a particular user to whom the current user want to message the current user is redirected to messaging interface page. Upon rendering, the component checks the user's authentication status, redirecting to the sign-in page if necessary. The main content of the page features a dynamic grid displaying chat messages between the current user and another user, with each message showing the sender's profile image, name, and the message content. There's also an input field for typing new messages and a "Send" button, enabling users to send messages in real-time.

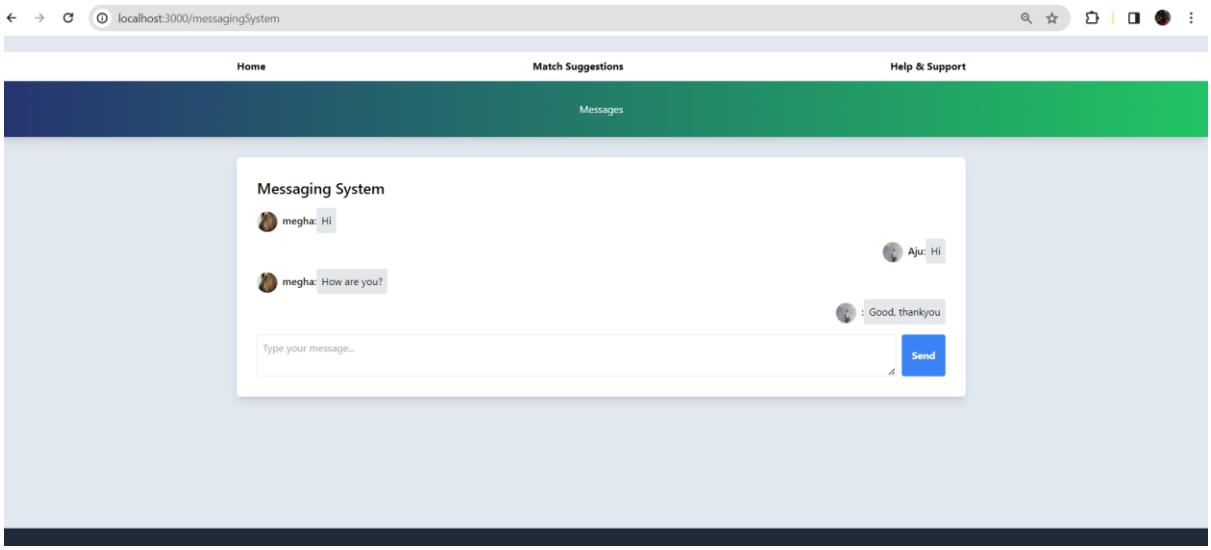


Figure 63: Messaging System UI

The component utilizes API calls to fetch the chat history from the backend server, updating the state accordingly. The MessagingSystem component showcases a well-structured and interactive messaging interface, combining both frontend and backend functionality for a seamless user experience.

Implementation code paths:

View:

- *Roommate-Matching-Platform\front-end\roommate-matching\pages\interface.js*
- *Roommate-Matching-Platform\front-end\roommate-matching\pages\messagingSystem.js*

Controller:

- *Roommate-Matching-Platform\back-end\rest\src\main\java\com\roommate\app\rest\controller\MessagingControllers.java*
- *Roommate-Matching-Platform\back-end\rest\src\main\java\com\roommate\app\rest\controller\RoommateControllers.java*
-

Model:

- *Roommate-Matching-Platform\back-end\rest\src\main\java\com\roommate\app\rest\Models\User.java*
- *Roommate-Matching-Platform\back-end\rest\src\main\java\com\roommate\app\rest\Repo\UserRepo.java*
- *Roommate-Matching-Platform\back-end\rest\src\main\java\com\roommate\app\rest\Models\Message.java*
- *Roommate-Matching-Platform\back-end\rest\src\main\java\com\roommate\app\rest\Models\RoomateSelection.java*
- *Roommate-Matching-Platform\back-end\rest\src\main\java\com\roommate\app\rest\Dto\RoommateSelectionDto.java*
- *Roommate-Matching-Platform\back-end\rest\src\main\java\com\roommate\app\rest\Dto\MessageDto.java*
- *Roommate-Matching-Platform\back-end\rest\src\main\java\com\roommate\app\rest\Dto\UserDto.java*

4.13 TESTIMONY

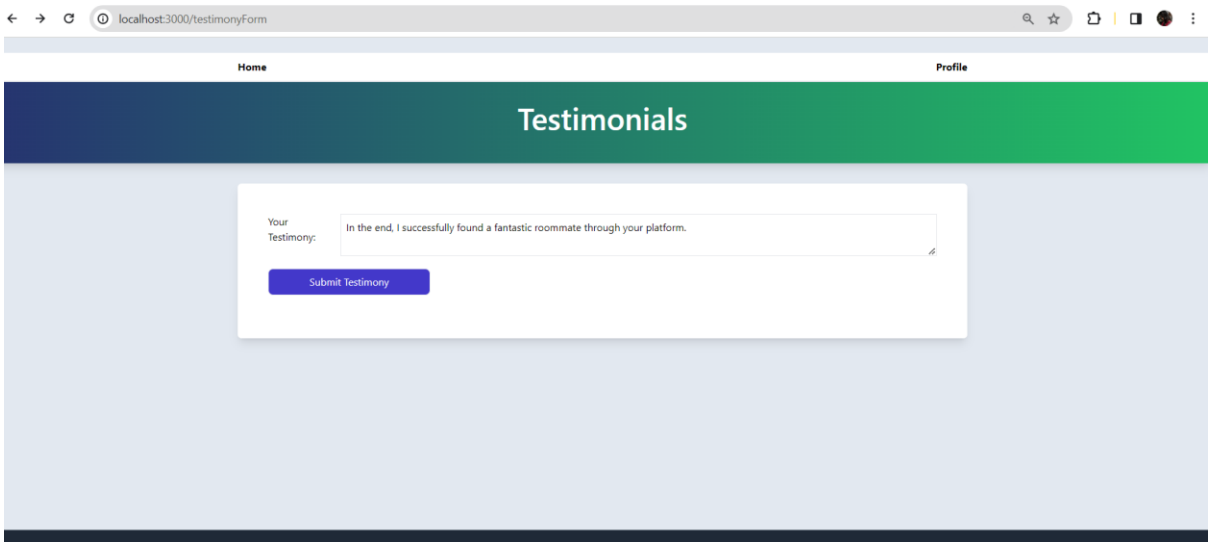


Figure 64: Testimony Page UI

The Testimony Form component in the application functions as a page where users can submit testimonials. Upon loading, the page verifies the user's authentication status by using the use Effect hook. When the user is not authorized, they are automatically routed to the sign-in page.

The Page has a submission form for testimonials, consisting of a text field where users may input their testimonies and a submit button.

```
const response = await fetch("http://localhost:8081/save-testimonials", {
  method: "POST",
  headers: {
    "Content-Type": "application/json",
    "Access-Control-Allow-Origin": "*",
  },
  body: JSON.stringify(formData), // Send the actual form data
});
```

Figure 65: API call to save the testimony

The *handleSubmit* method processes the submitting data by making a POST request to the server's *save-testimonials* endpoint, including the user ID and testimony content to save the data into database. When a submission is successful, it will cause a redirect to the "*aboutUs*" page after a certain wait. On the other hand, if any problems occur, they will be recorded in the console. The testimonial added by the user can be visible in *about-us* page.

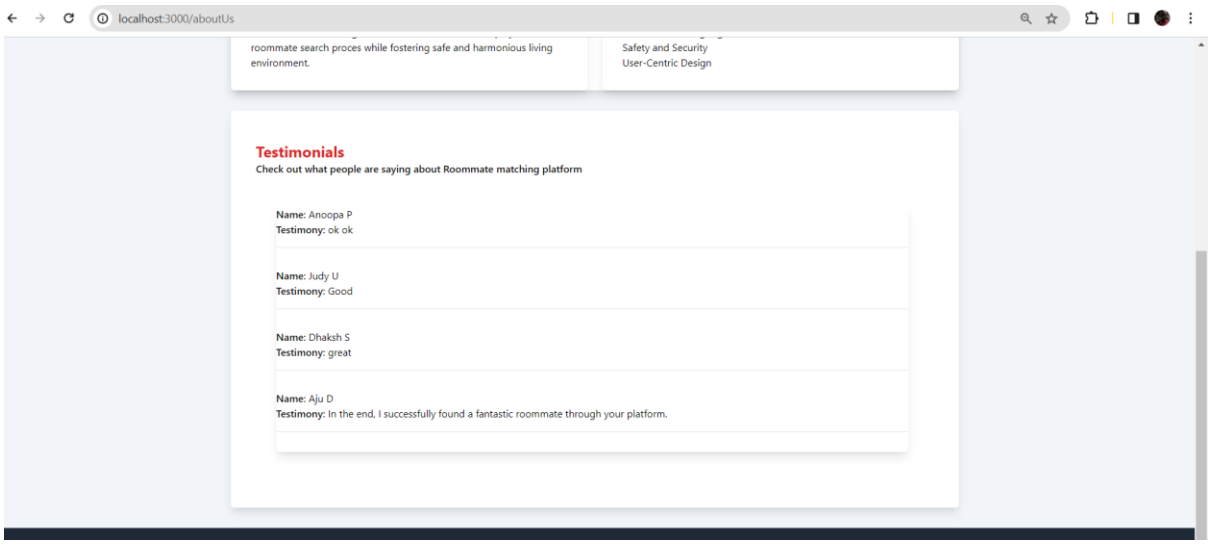


Figure 66: Testimonial of the user successfully saved

Implementation code paths:

View:

- *Roommate-Matching-Platform\front-end\roommate-matching\pages\testimonyForm.js*

- *Roommate-Matching-Platform\front-end\roommate-matching\pages\aboutUs.js*

Controller:

- *Roommate-Matching-Platform\back-end\rest\src\main\java\com\roommate\app\rest\controller\OtherApiControllers.java*
- *Roommate-Matching-Platform\back-end\rest\src\main\java\com\roommate\app\rest\controller\ApiControllers.java*
-

Model:

- *Roommate-Matching-Platform\back-end\rest\src\main\java\com\roommate\app\rest\Models\User.java*
- *Roommate-Matching-Platform\back-end\rest\src\main\java\com\roommate\app\rest\Repo\UserRepo.java*
- *Roommate-Matching-Platform\back-end\rest\src\main\java\com\roommate\app\rest\Dto\UserDto.java*
- *Roommate-Matching-Platform\back-end\rest\src\main\java\com\roommate\app\rest\Repo\TestimonialsRepo.java*
- *Roommate-Matching-Platform\back-end\rest\src\main\java\com\roommate\app\rest\Model\Testimonials.java*
- *Roommate-Matching-Platform\back-end\rest\src\main\java\com\roommate\app\rest\Dto\TestimonialDto.java*

4.14 FEEDBACK AND RATING

Similar to testimony page Feedback and rating can also be provided by the users the images of the UI implemented are as shown in below:

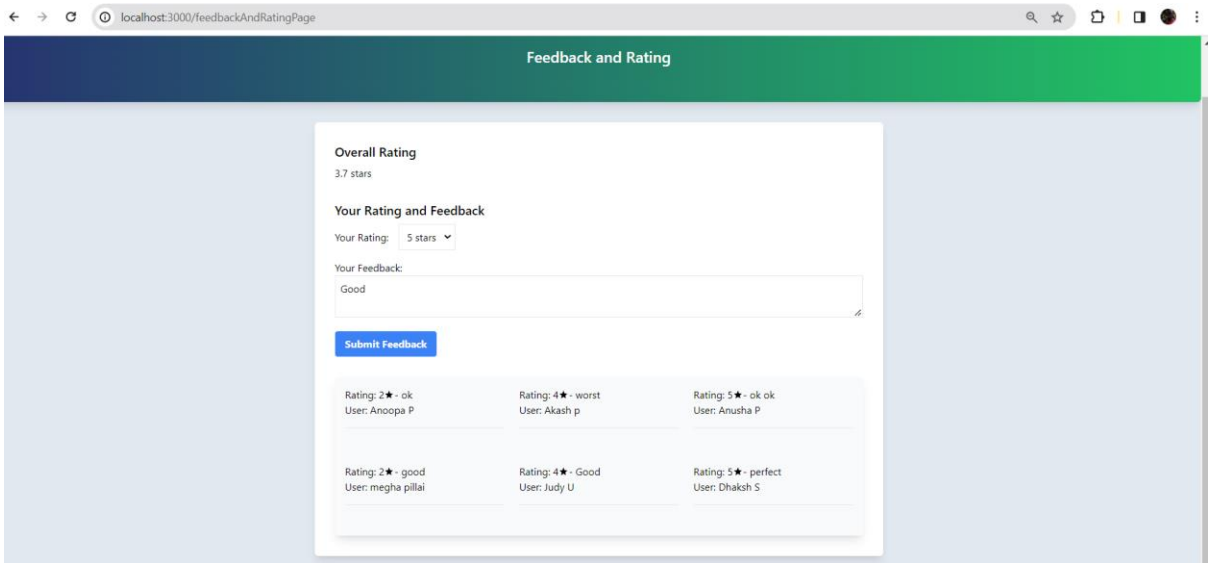


Figure 67: Feedback and Rating page UI

```
const fetchRatingsData = async () => {
  try {
    const userId = jsCookie.get("user") || "";
    const response = await fetch(
      `http://localhost:8081/get-ratings?userId=${userId}`,
      {
        headers: {
          "Content-Type": "application/json",
          "Access-Control-Allow-Origin": "*",
        },
      }
    );
```

Figure 68: API call to get ratings by all the application users

View:

- *Roommate-Matching-Platform\front-end\roommate-matching\pages\feedbackandRatingPage.js*

Controller:

- *Roommate-Matching-Platform\back-end\rest\src\main\java\com\roommate\app\rest\controller\OtherApiControllers.java*
- *Roommate-Matching-Platform\back-end\rest\src\main\java\com\roommate\app\rest\controller\ApiControllers.java*
-

Model:

- *Roommate-Matching-Platform\back-end\rest\src\main\java\com\roommate\app\rest\Models\User.java*
- *Roommate-Matching-Platform\back-end\rest\src\main\java\com\roommate\app\rest\Repo\UserRepo.java*
- *Roommate-Matching-Platform\back-end\rest\src\main\java\com\roommate\app\rest\Dto\UserDto.java*
- *Roommate-Matching-Platform\back-end\rest\src\main\java\com\roommate\app\rest\Repo\RatingsRepo.java*
- *Roommate-Matching-Platform\back-end\rest\src\main\java\com\roommate\app\rest\Model\Rating.java*
- *Roommate-Matching-Platform\back-end\rest\src\main\java\com\roommate\app\rest\Dto\RatingDto.java*

4.15 MATCHING SUGGESTION

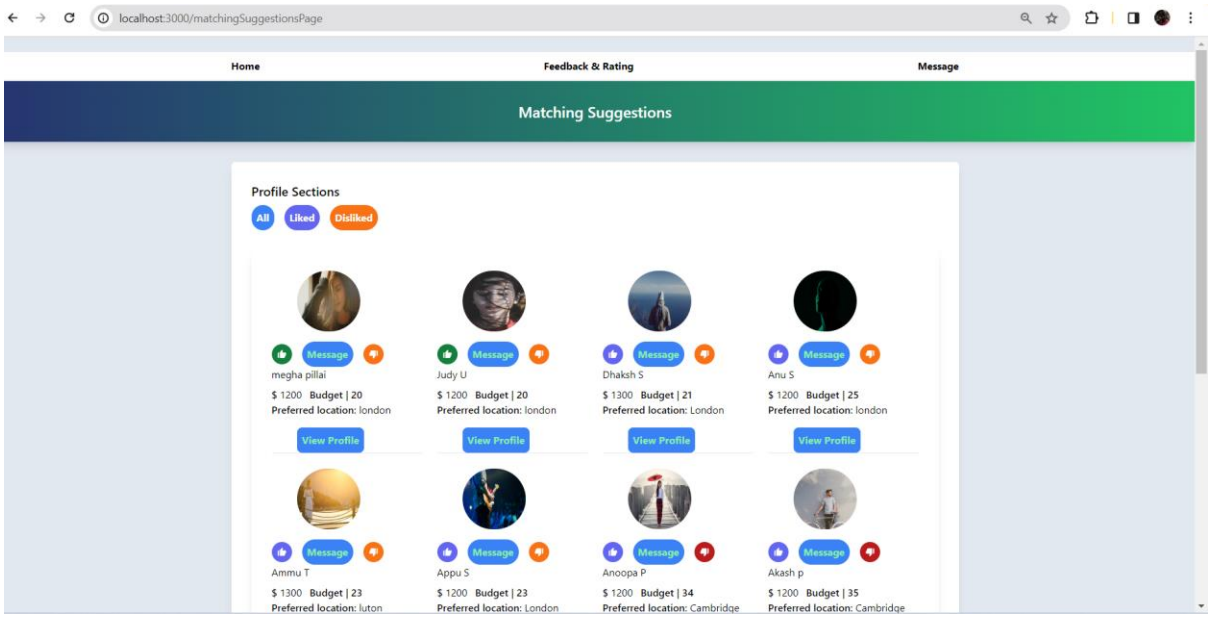


Figure 69: Matching Suggestion Page UI

The *MatchingSuggestionsPage* is a page for listing roommate matching and a feature to like dislike the profiles, including several essential features. During the startup process, the code verifies user authentication by examining the presence of a valid user ID in cookies. If authentication is not confirmed, the code sends the user to the sign-in page. Afterwards, it retrieves the verified user's profile information from the server by making an API request to *http://localhost:8081/profile/${userId}*. The data, including information such as name, budget, age, and desired location, is thereafter presented on the website.

The component retrieves roommate ideas by performing an API connection to *http://localhost:8081/api/roommates/matches-compatiblility/$userId* to determine compatibility. The proposals are shown in a grid format, with each suggestion followed with choices to express approval, disapproval, and send a note to the corresponding roommate. The code utilizes local state management to handle user preferences, enabling dynamic modifications without the need for a page refresh.

In addition, the component retrieves profile photographs for each roommate by making API requests to *http://localhost:8081/profile-picture/$roommate.id.* The photographs are shown with the roommate profiles on the grid, augmenting the user experience.

In order to enhance the user experience, the code incorporates the capability of section filtering, enabling users to choose display profiles according to categories such as "Liked," "Disliked," or "All." This feature improves user navigation and increases user engagement with the Roommate matching process.

The code implemented has features for liking and disliking recommended roommates, and it communicates with the server through two API endpoints:

For liking a profile:

- API endpoint: http://localhost:8081/api/roommates/likeHTTP
- Method: POST
- Objective: This API is called when a user likes a roommate. The client initiates a POST request including essential data such as the ID of the current user, the ID of the person being liked, the list of preferred roommates, and the list of disliked roommates. The server handles this information, modifies the database, and provides a corresponding response.

For disliking profiles:

- URL: http://localhost:8081/api/roommates/dislike
- HTTP Method: POST (to add a hate) / DELETE (to delete a dislike)
- Purpose: This endpoint is utilized when a user dislikes a roommate suggested. Just like the like endpoint, the client uses a POST request to add a dislike and a DELETE request to remove a dislike. The server handles this data, modifies the database, and provides a corresponding response.

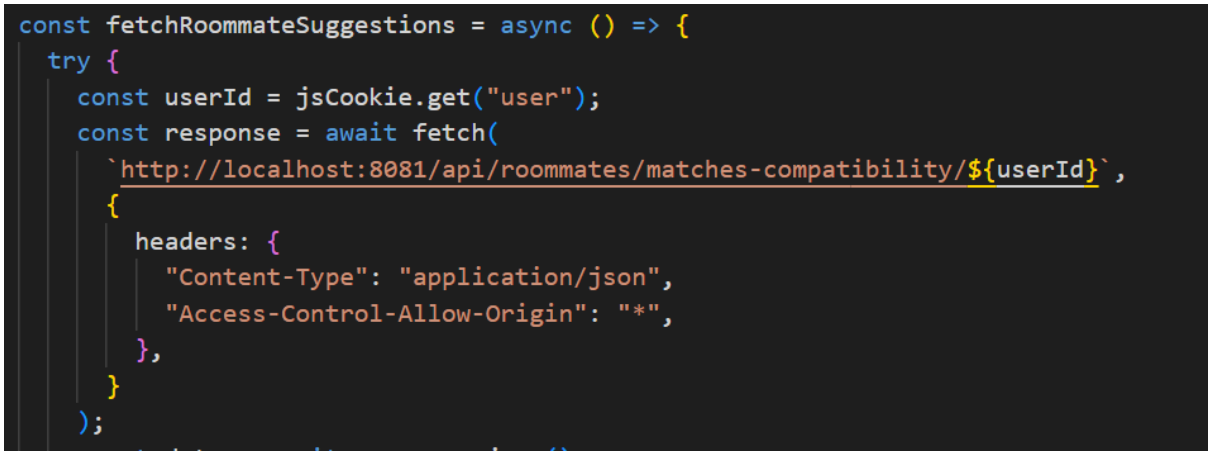


Figure 70: API call to get roommate compatibility.

The "View Profile" button in the code functions as a user interface component that enables people to see the comprehensive profile view of a certain roommate. The button is presented selectively for each suggested roommate and initiates a navigation event upon being clicked. Upon clicking the button, users are sent to the profile page of a designated roommate, where they can see detailed profile of the user profile the current user have selected.

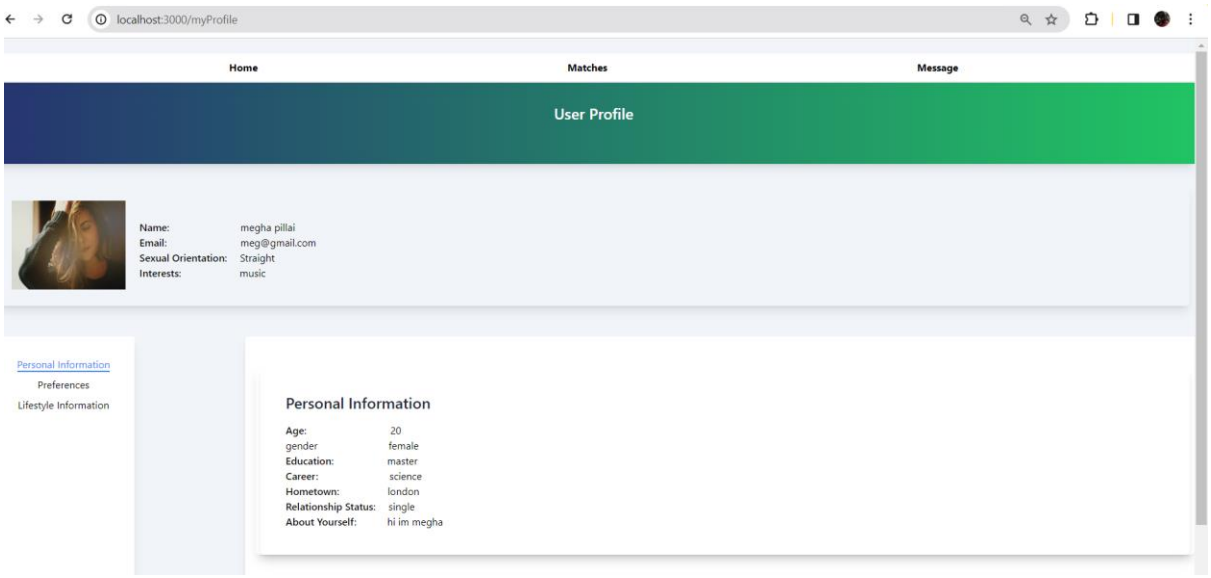


Figure 71: Potential Roommate Profile view

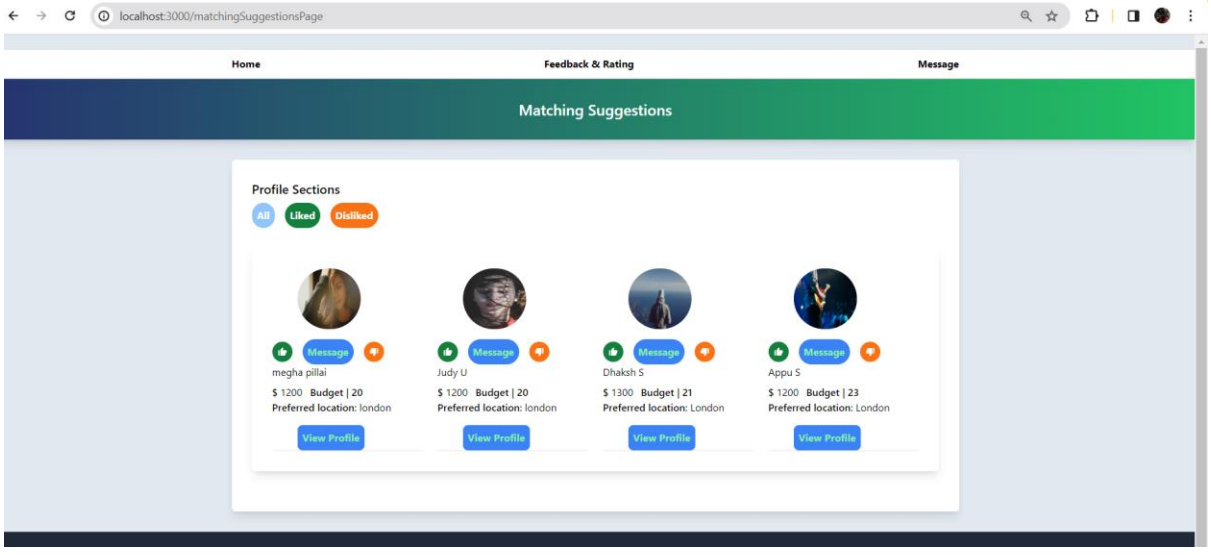


Figure 72: Like profiles view

```
await fetch(`http://localhost:8081/api/roommates/like`, {
  method: "POST",
  headers: {
    "Content-Type": "application/json",
    "Access-Control-Allow-Origin": "*",
  },
  body: JSON.stringify({
    liker: jsCookie.get("user"),
    liked: roommateId,
    likedRoommates: updatedLikedRoommates,
    dislikedRoommates,
  }),
});
```

Figure 73: API call to add liked profiles

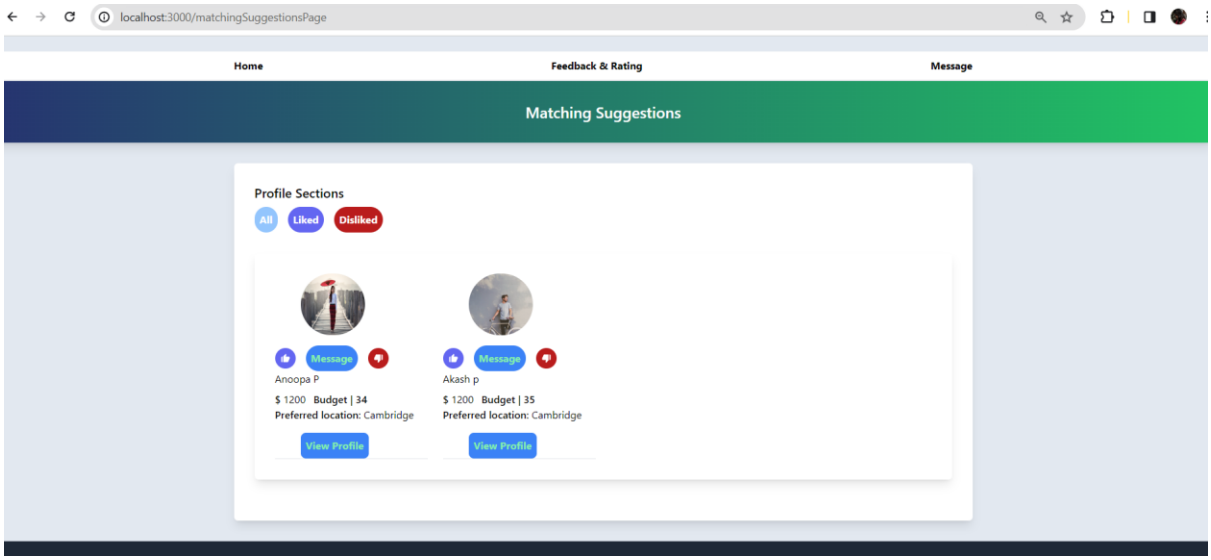


Figure 74: Dislike profiles view

```
  await fetch(`http://localhost:8081/api/roommates/dislike`, {
    method: "POST",
    headers: {
      "Content-Type": "application/json",
      "Access-Control-Allow-Origin": "*",
    },
    body: JSON.stringify({
      liker: jsCookie.get("user"),
      liked: roommateId,
      likedRoommates,
      dislikedRoommates: updatedDislikedRoommates,
    }),
  });
}
```

Figure 75: API call to add disliked profiles

View:

- *Roommate-Matching-Platform\front-end\roommate-matching\pages\feedbackandRatingPage.js*

Controller:

- *Roommate-Matching-Platform\back-end\rest\src\main\java\com\roommate\app\rest\controller\OtherApiControllers.java*
- *Roommate-Matching-Platform\back-end\rest\src\main\java\com\roommate\app\rest\controller\ApiControllers.java*
- *Roommate-Matching-Platform\back-end\rest\src\main\java\com\roommate\app\rest\controller\RoommateControllers.java*
- *Roommate-Matching-Platform\back-end\rest\src\main\java\com\roommate\app\rest\controller\MessagingContoller.java*

Model:

- *Roommate-Matching-Platform\back-end\rest\src\main\java\com\roommate\app\rest\Models\User.java*
- *Roommate-Matching-Platform\back-end\rest\src\main\java\com\roommate\app\rest\Repo\UserRepo.java*
- *Roommate-Matching-Platform\back-end\rest\src\main\java\com\roommate\app\rest\Dto\UserDto.java*
- *Roommate-Matching-Platform\back-end\rest\src\main\java\com\roommate\app\rest\Repo\MessageRepo.java*
- *Roommate-Matching-Platform\back-end\rest\src\main\java\com\roommate\app\rest\Model\Message.java*
- *Roommate-Matching-Platform\back-end\rest\src\main\java\com\roommate\app\rest\Dto\MesaageDto.java*

- *Roommate-Matching-Platform\back-end\rest\src\main\java\com\roommate\app\rest\Repo\RoommateSelectionDto.java*
- *Roommate-Matching-Platform\back-end\rest\src\main\java\com\roommate\app\rest\Model\RoommateSelection.java*
- *Roommate-Matching-Platform\back-end\rest\src\main\java\com\roommate\app\rest\Dto\RoommateSelection.Dto.java*

4.16 LOGOUT

The *handleLogout* method is responsible for managing the process of logging out in the given code. When called, it utilizes the *jsCookie.remove*("*user*") command to eliminate the user authentication cookie ("user"). Afterwards, the function uses the router to guide the user to the "/sign-in" page, so redirecting them to the sign-in or login page. This approach efficiently logs out the user from the application by removing their authentication credentials and redirecting them to the sign-in page.

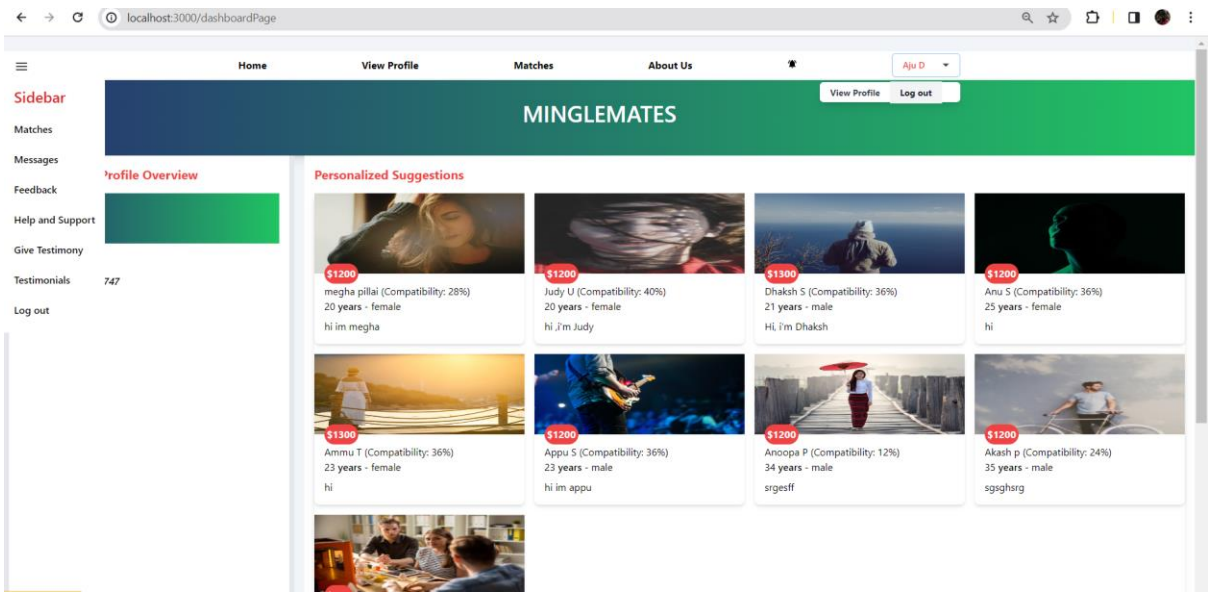


Figure 76: Logout option In Dashboard

```
<li>
  <Link href="/#" onClick={handleLogout}>
    <span className="block py-2 hover:underline">Log out</span>
  </Link>
</li>
```

Figure 78: Logout button code

```
const handleLogout = () => {
  jsCookie.remove("user");
  router.push("/sign-in");
};
```

Figure 79: Logout method

View:

- *Roommate-Matching-Platform\front-end\roommate-matching\pages\dashboardPage.js*

5. QUALITY ASSURANCE AND PERFORMANCE ASSESSMENT

5.1 API TESTING

Server Check

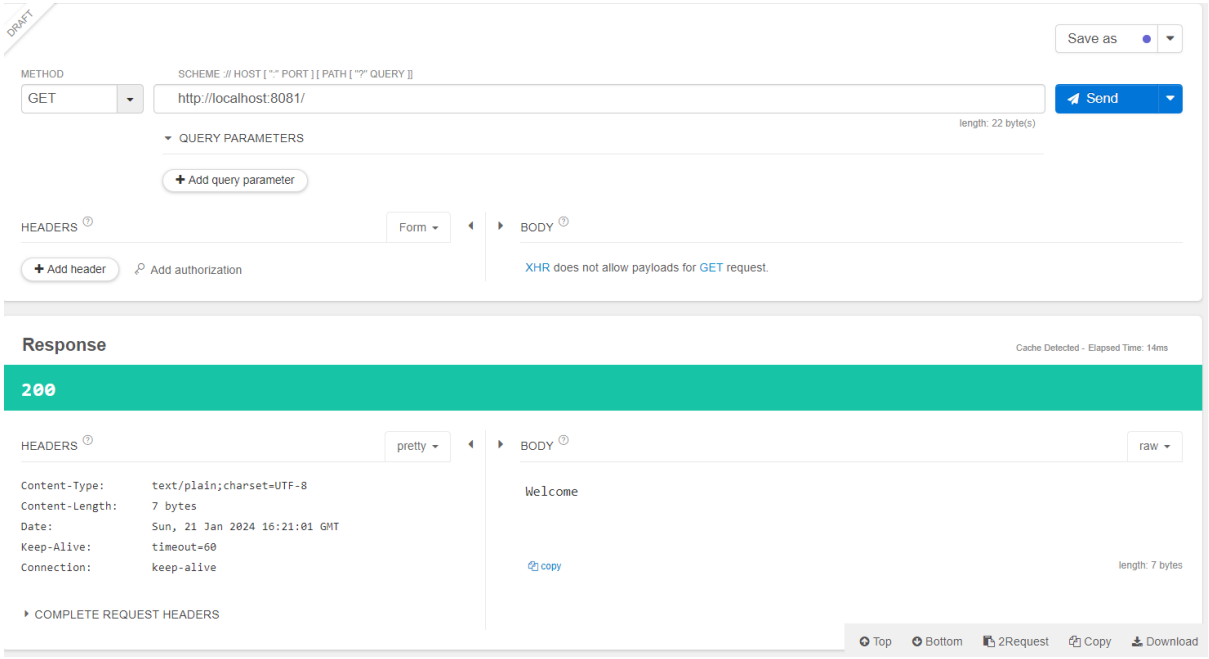


Figure 80: API testing for server check

This API call was performed to ascertain the functional state of the server, affirming its promptness in responding. The request is targeted at the endpoint '/', and when this API is invoked, the server-side code handles the request and provides a response. The matching response can be observed in the BODY part of the illustrated figure.

Test Results:

The API call successfully obtained the required response.

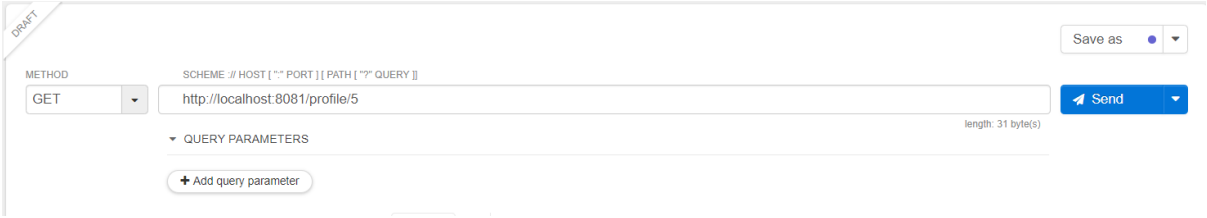
API call for getting all the details of a user

The figure depicts the API request performed to fetch the profile details of a particular user. The goal of this request was to retrieve detailed data about the user identified by UID =5 from the server. The picture also depicts the subsequent response given by the server after processing

the API call. The response includes detailed user profile information, including fields such as name, email address, lifestyle information, and more.

Test Results:

The API call successfully retrieved the necessary response.




```
    gender: "male",
    education: "sdfsdf",
    career: "srgsg",
    hometown: "svsefef",
    relationshipStatus: "single",
    aboutYourself: "srgesff",
    userId: null
},
lifestyleInformationDto: {
    cleanliness: "Neat freak",
    workSchedule: "Full-time",
    sleepingHours: "Early riser",
    tobacco: "Smoke regularly",
    alcohol: "Social drinker",
    drugs: "Non user",
    roommateInteraction: "Close friends",
    entertainFriendsFrequency: "Rarely",
    cookingHabits: "Gourmet chef",
    describeYou: "Military",
    userId: null
},
user: {
    id: 5,
    firstName: "Anoopa",
    lastName: "P",
    phoneNumber: "123456789",
```


Figure 90: API call and response for getting a user profile details

API call for getting testimonials.

An API call was initiated to obtain testimonials from all users using the following endpoint: *http://localhost:8081/get-testimonials*

Upon invocation of this API, the server undertakes the task of processing the request to collect all testimonials and their corresponding information, which encompass the usernames of the users who have provided each individual testimony. Afterwards, the server transmits a response, the contents of which are exhibited in the BODY part of the figure.

Test Results:

The API call successfully retrieved the necessary response.

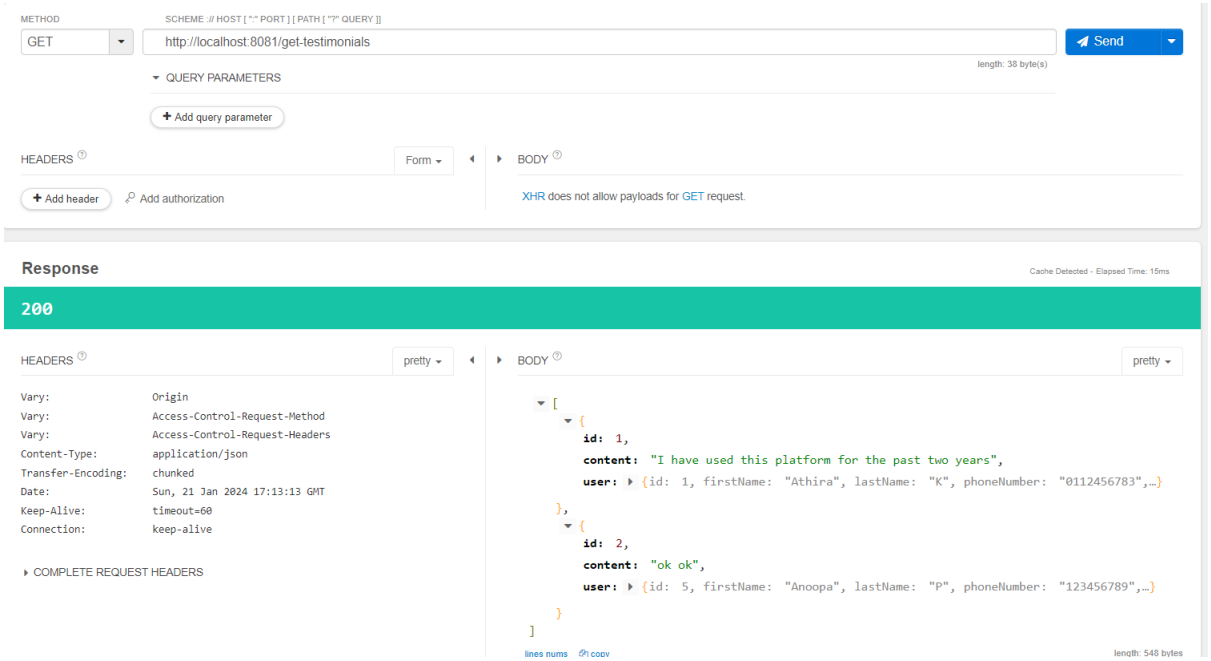


Figure 91: API call and response for getting testimonials

API call for getting compatibility score.

To determine the compatibility % between the user with UID 5 and all other users, an API request was called using the endpoint *http://localhost:8081/api/roommates/matches-compatibility/4*. Upon receiving this API call, the server does computations to ascertain the compatibility score of the user with UID 5 in relation to all other users. Subsequently, the server produces a response that includes a list of all users, along by their particulars and associated compatibility percentages, as seen in the diagram.

Test Results:

The API call successfully retrieved the necessary response.

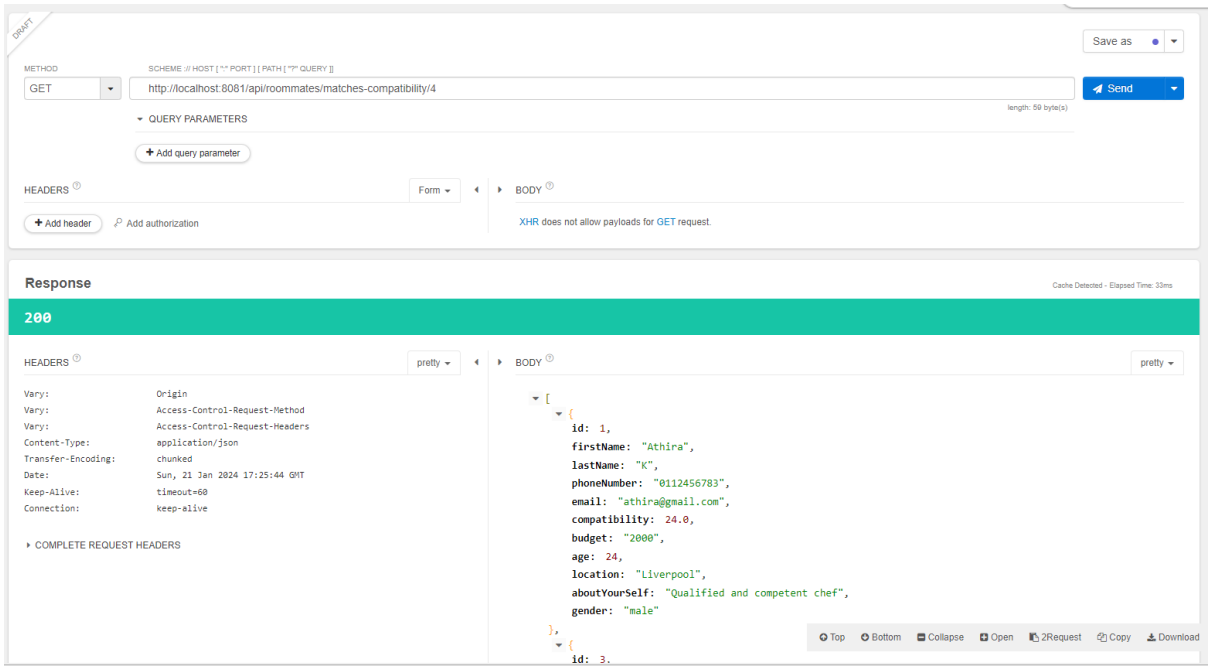


Figure 92: API call and response for getting compatibility score

API call for user registration.

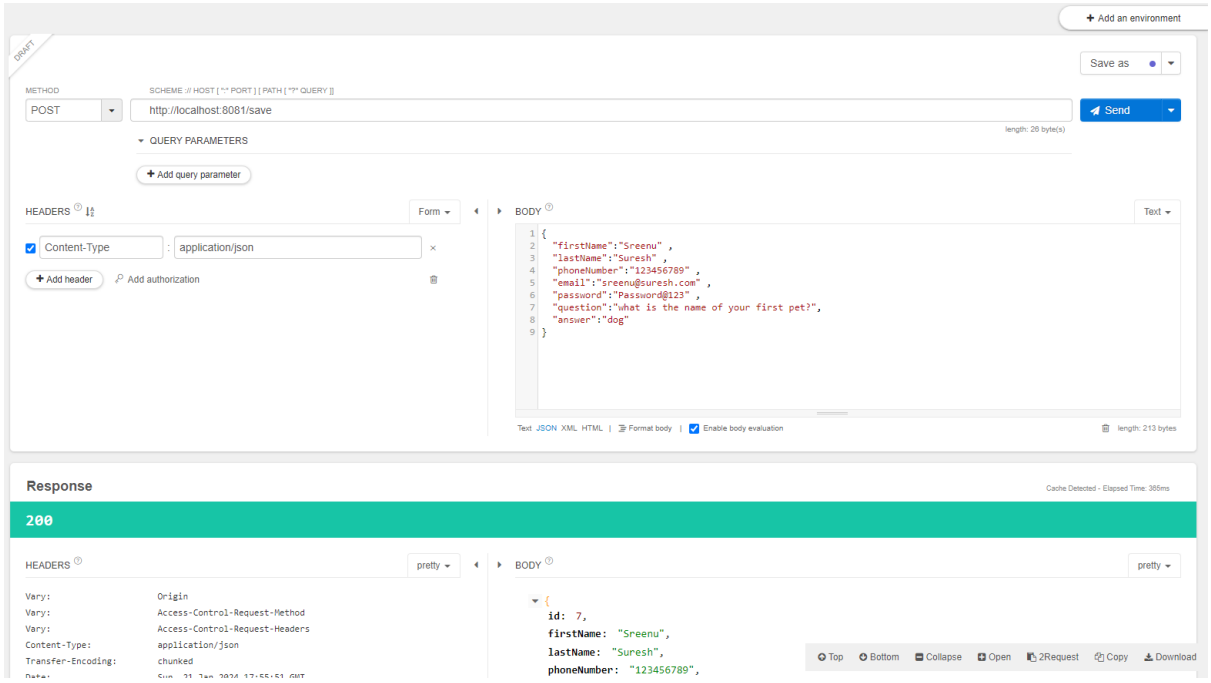


Figure 93: API call and response for user registration

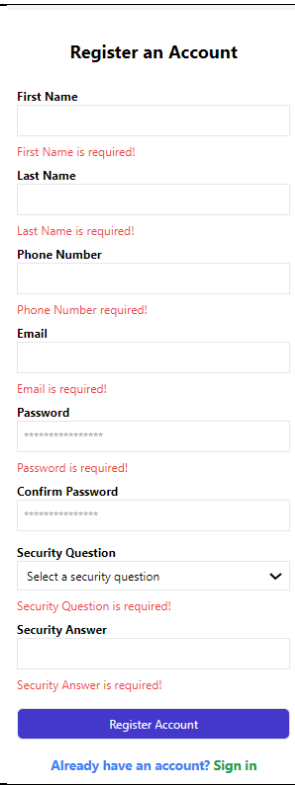
The API request made to the endpoint *http://localhost:8081/save* is a POST request specifically intended for user registration. The user registration information is transmitted in the request body of this request, using the JSON format. The server analyses the data, handles the information from the POST request, records the user in the database, and provides a response that includes the UID (User ID) created for the newly registered user which can be seen in the BODY section of the response from the figure.

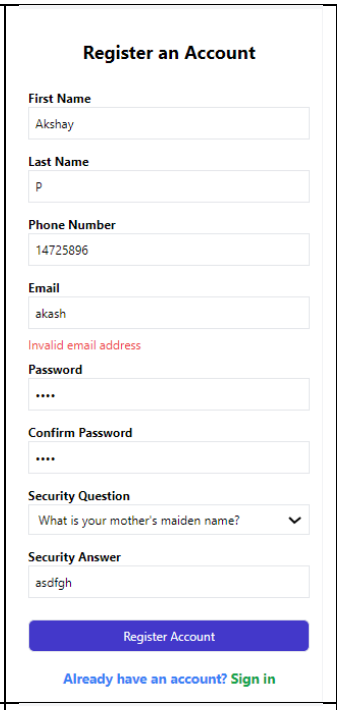
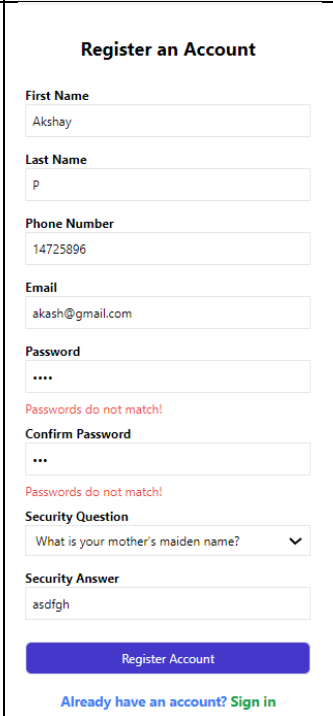
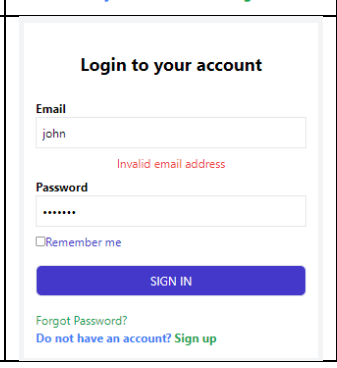
Test Results:

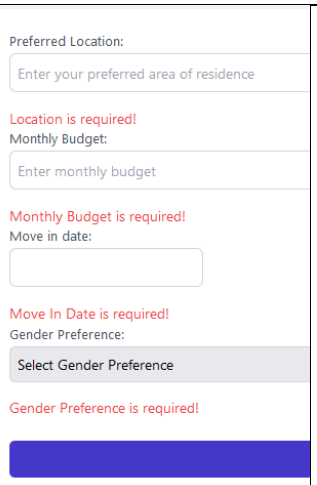
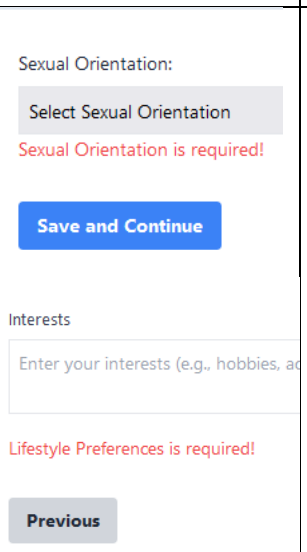
The API call successfully retrieved the necessary response.

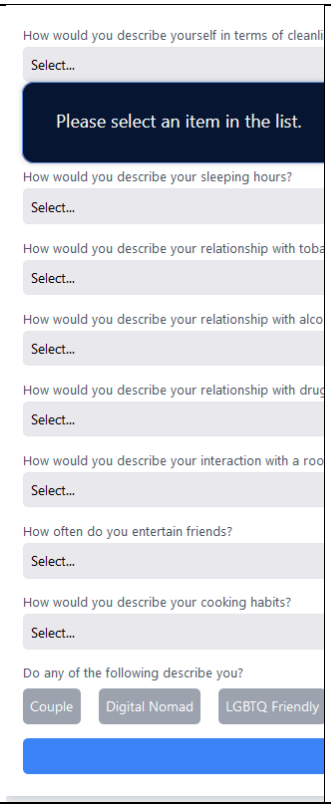
5.2 TESTING FORM VALIDATION

Form validation is an essential component of web development that guarantees the completeness of user inputs. The provided test cases focus on various aspects of user functionality, including registration, login, preferences submission, profile details, and lifestyle information. In all scenarios, the desired result is to block the submission of the form and notify the user to rectify any missing or improperly formed data.

FORM	TEST CASE	TEST OUTPUT	REMARKS
REGISTER	Trying to register the user without filing one or more fields. Expected Output: Form should not be submitted. Should alert the user all the required fields.		Test case worked as per expectations all the fields of the form are necessary and the form cannot be submitted with one or more fields empty

REGISTER	Wrong Email format Expected Output: The form should not be submitted. Alert the user		Test case worked as per expectations the form cannot be submitted when correct email format is not followed.
REGISTER	Password and Confirm Password do not match. Expected Output: The form should not be submitted. Alert the user		Test case worked as per expectations the form cannot be submitted when Password and Confirm Password do not match
LOGIN	Wrong Email format Expected Output: The form should not be submitted. Alert the user		Test case worked as per expectations the form cannot be submitted when correct email format is not followed.

PREFRENCES	Trying to submit preferences of the user without filing one or more fields. Expected Output: Form should not be submitted. Should alert the user all the required fields.		Test case worked as per expectations all the fields of the form are necessary and the form cannot be submitted with one or more fields empty
PROFILE	Trying to submit profile details of the user without filing one or more fields. Expected Output: Form should not be submitted. Should alert the user all the required fields.		Test case worked as per expectations all the fields of the form are necessary and the form cannot be submitted with one or more fields empty

LIFESTYLE INFO	Trying to submit lifestyle info of the user without filing one or more fields. Expected Output: Form should not be submitted. Should alert the user all the required fields.		Test case worked as per expectations all the fields of the form are necessary and the form cannot be submitted with one or more fields empty

5.3 FUNCTIONAL TESTING

Functional testing is an essential component of the quality assurance process, guaranteeing that every module in the Minglemates functions as intended and provides a smooth user experience. This testing step seeks to ensure that the platform fits the defined objectives by carefully analyzing the functions of important modules such as Registration, Login, Profile Customization, Preferences, Matching Roommates, Messaging, Feedback, Testimony, and Logout. Every module, from user onboarding to communication, lifestyle customization to roommate matching, and user feedback to logout processes, is thoroughly examined to detect any deviations from the anticipated behavior.

REGISTER MODULE

Test case: The user must correctly fill out the registration form. Upon form submission, the user's registration should be confirmed, and the user be directed to the login page.

Expected Output: The user should be able to fill out the registration form. Upon form submission, the user's registration should be confirmed, and the user should be directed to the login page.

.

Test Outcome: The user successfully filled the signup form. The user successfully submitted their information by complying with all validation requirements and accurately completing the form. Afterwards, the user's registration was verified, and the webpage redirected to the login page.

Remarks: The test case ran without any issues, demonstrating the successful implementation of the Register module. Users could easily complete the registration procedure, and after they submit their information, they are immediately registered and sent to the login page. This confirms the effective creation and operation of the User Registration module.

LOGIN MODULE

Test Case: The user provides accurate login credentials in the login form. After a successful login, if it is the user's initial login, they must configure their profile and preferences before being able to access the application dashboard. Upon future logins, the user should be automatically sent to the dashboard page. The user should have the capability to reset their password by utilizing the "Forgot Password" feature. After a successful login, sessions should be established in the frontend utilizing cookies. If the credentials used for login are incorrect then user should be notified.

Expected Outcome: The user successfully authenticates by entering the accurate login information. For the initial login, the user establishes their profile and preferences. Upon subsequent logins, the user is immediately sent to the dashboard. The password reset feature is both accessible and operational. Sessions are established using cookies following a successful login. For wrong credentials user should be notified.

Test Outcome: The user successfully authenticates using the right login credentials. During the initial login, the user establishes their profile and preferences. Upon subsequent logins, the user is immediately sent to the dashboard. The "Forgot Password" feature enables users to initiate the process of resetting their password. Frontend cookies are utilised to generate sessions effectively. During invalid credentials usage for login user is notified with failed to sign in message.

Remarks: The test case was successfully completed, validating the anticipated functionality of the Login module. Users can authenticate using accurate login credentials, establish profiles and settings on their initial login, and effortlessly access the dashboard during subsequent

logins. The "Forgot Password" feature and cookie-based sessions function as designed and developed, hence improving the overall user experience.

PROFILE MODULE

Test case: Upon first login, the user should have the capability to establish their profile, which encompasses the addition of a profile photo, interests, and sexual orientation. During the initial login, the user should have the capability to provide personal details, including age, gender, education, profession, place of origin, relationship status, and a concise self-description. Upon successive logins, the user should possess the capability to alter and revise their personal information.

Expected Output: Upon the initial login, the user successfully sets up a profile, encompassing a profile photo, hobbies, and sexual orientation. During the initial login, the user provides personal details such as age, gender, education, occupation, place of origin, relationship status, and a self-description. Subsequent logins enable the user to alter and refresh their personal information.

Test Outcome: Upon the initial login, the user successfully establishes a profile by providing a profile photo, indicating their hobbies, and specifying their sexual orientation. During the initial login, the user provides precise personal information. Upon subsequent logins, users has the capability to change and update their personal information.

Remarks: The test case was successfully completed, validating the correct operation of the Profile module. Upon initial login, users was able to effortlessly establish their profiles, incorporating a range of particulars, and subsequently revise this data during subsequent logins. This guarantees a dynamic and customized user experience on the platform.

FEEDBACK MODULE

Test case: The user should have the capability to modify the rating and the feedback that was previously submitted. When modifying the rating and the feedback, the user should have the ability to revise both the rating and the feedback.

Expected Output: The user has successfully submitted a rating and the feedback for the application. The user has the opportunity to modify the rating and the feedback that was

previously given. By modifying the rating and the feedback, the user has the ability to modify both the rating and the feedback.

Test Outcome: The user offers a rating and the feedback for the application in an effective manner. The application enables the user to modify the rating and the feedback that was previously submitted. During the editing process, the user effectively modifies both the rating and the feedback.

Remarks: The test case was completed without any issues, validating the correct operation of the Feedback module. Users could conveniently rate the application and subsequently modify these ratings, so changing both the rating and the accompanying feedback. This adaptability guarantees that consumers could consistently enhance their evaluations of the application.

TESTIMONIAL MODULE

Test case: The user provides a testimonial on their experience using the application. The user should possess the capability to modify the testimonial that was previously submitted. When modifying the testimonial, the user should have the capability to revise the material.

Expected output: The user has successfully submitted a testimonial detailing their experience. The user has the opportunity to modify the testimonial that was previously submitted. By changing the testimonial, the user has the ability to modify the content.

Test Outcome: The user offers a testimonial on their experience on the platform. The technology enables the user to modify the testimonial that was previously submitted. During the editing process, the user effectively modifies the content of the testimonial.

Remarks: The test case was successfully completed, verifying the accurate operation of the Testimonial module. Users may effortlessly submit testimonials on their experiences and subsequently modify these testimonials, making updates to the material as necessary. This guarantees that users are able to express their changing ideas about the site

MESSAGING MODULE

Test case: The user sends a message to another user. The user experiences seamless reception of messages from other users. The chat interface should be consistently accessible from the

beginning of the conversation. The user interface should provide the recipient's message on the right side and the sender's message on the left side.

Expected Output: The user is able to send messages to other users without any complications. The user may effortlessly receive messages from other users. The chat interface remains available during the whole discussion. The user interface accurately displays the recipient's message on the right side and the sender's message on the left side.

Test Outcome: The user successfully begins communication with other users. Incoming messages from other users are received seamlessly. The chat interface is continuously accessible during the whole conversation. The user interface effectively presents the recipient's message on the right side and the sender's message on the left side.

Remarks: The test case was completed without any issues, validating the correct operation of the Messaging module. Users may communicate seamlessly without any problems, and the chat interface remains accessible from the beginning of the session. The UI display adheres to the anticipated arrangement, hence boosting the user experience during interactions.

MATCHING ROOMATE MODULE

Test case: After login in, the dashboard should display comprehensive information about all current users, including their compatibility percentages. By selecting a specific user in the dashboard, the system should present a detailed profile of that person. Furthermore, the matches recommendation page should offer convenient access to the chatting feature.

Expected Output: Upon each login, the dashboard should display information about all other current users, including their compatibility %. Selecting a specific user from the dashboard should result in the presentation of that person's comprehensive profile. The message option should be accessible on the matches suggestion page.

Test outcome: The user gets presented with an extensive dashboard that displays precise information about existing users and their compatibility percentages. Users were routed to a profile page that displayed crucial information about the selected individual. By clicking on the message option, the user was seamlessly transferred to the messaging page.

Remark: The test case guarantees a smooth and user-friendly experience in the Matching Roommate Module. The anticipated results, which include the display of user information and

compatibility percentages on the dashboard, the accessibility of profiles from the dashboard, and the availability of messaging options, all contribute to a strong and efficient platform.

To summarize, the thorough examination of every module in the platform confirms that all features perform flawlessly in accordance with their intended requirements. The Registration Module efficiently takes and processes user data, guaranteeing a seamless onboarding experience. The Login Module offers a secure and customized access point, improving user security and providing personalized experiences. The Profile Module enables users to customize their profiles, promoting a feeling of uniqueness and allowing more accurate roommate matching. The Preferences Module greatly enhances user experience by enabling consumers to specify their lifestyle preferences, hence impacting compatibility evaluations. The Matching Roommate Module utilizes a sophisticated algorithm to accurately calculate compatibility percentages, resulting in relevant and precise roommate options. The Messaging Module provides a dependable and effective communication platform, enabling users to communicate and interact easily. The Feedback Module and Testimony Module offer users the opportunity to share their experiences, therefore contributing to the ongoing enhancement of the platform.

5.4 PERFORMANCE EVALUATION

The application had a thorough testing process, which covered a complete assessment of several features, such as API functioning, form validation, and overall system operation. The subsequent sections offer a thorough evaluation of the testing procedure and an analysis of the application's performance.

Testing of Application Programming Interfaces (APIs):

- API Call for Server Check: The test for server check confirmed that the application can establish successful communication with the server, verifying its availability and responsiveness.
- API Call to Retrieve All User Details: The test successfully obtained user details, verifying the capability of accessing and presenting user information.
- The API testing for retrieving testimonials assessed the application's ability to successfully obtain and exhibit user testimonials according to the expected functionality.

- The compatibility score API call has verified the application's ability to appropriately compute and report compatibility scores.
- API Call for User Registration: The test for user registration verified that new users are able to register successfully, confirming the correct operation of the registration process.

Form Validation:

- Registration Form Validation: The validation tests for the registration form have shown that the registration process complies with predetermined rules, guaranteeing the integrity and correctness of the data.
- Login Form Validation: The login form validation tests ensured the secure and precise verification of user credentials during the login procedure.
- Profile Setup Form Validation: The purpose of profile setup form validation was to verify that users entered precise and legitimate information while creating their profiles.
- Validation of the preferences setup form has verified that users are able to configure their preferences in accordance with the application's specifications.
- The validation process of the lifestyle information setup form ensures that users input precise lifestyle information, hence enhancing the application's customized experience.

Functionality Testing:

Module Functionality Testing: Thorough testing of all application modules indicated that the functionality is in line with the anticipated results. All deviations were identified and fixed throughout the testing process.

5.4.1 OVERALL PERFORMANCE EVALUATION AND RECCOMENDATIONS

Upon the successful completion of the specified tests, it could be concluded that the application accomplishes its design standards and operates as intended. The testing method successfully detected and corrected any flaws, guaranteeing a strong and dependable application. Although the testing phase yielded positive results, it is necessary to implement ongoing monitoring and regular testing to proactively identify and resolve any possible issues that may emerge throughout the application's development. Furthermore, doing performance testing with different loads and situations helps enhance the application's scalability and stability.

6. CONCLUSION

In the pursuit of optimizing shared living arrangements, this project has been defined by a set of explicit aims and objectives. The main objective of Minglemates was to create a platform that not only simplifies the process of finding suitable roommates but also goes beyond traditional models by promoting long-lasting relationships and providing a satisfying communal living experience.

At the forefront of this project goals was the creation and implementation of a matching algorithm. This computational architecture, guided by lifestyle choices, personal interests, and room preferences, seeks to not only guarantee compatibility but also alter the traditional concept of roommate matching. The goal was to carefully coordinate a complex arrangement of living together that aligns with the unique preferences of every individual.

The second objective highlights the crucial significance of an Intuitive User Interface (UI). The focus lies on the flawless integration of visually appealing and simple navigation, resembling a well-organized application UI. Effective communication is crucial for coordinating and balancing shared life experiences. The platform facilitates meaningful interaction between users by incorporating strong communication elements, which plays a crucial role in fostering a sense of commonality among housemates.

The development of Minglemates was carried out with careful attention to detail, utilizing the powerful features that allow React.js and Spring Boot. React.js, selected for its declarative and component-based methodology, enabled the development phase to create a dynamic and responsive user experience. Java Spring Boot offered several advantages on the backend. Spring Boot, known for its reputation in terms of dependability, scalability, and seamless integration, was important in the development process by enabling the creation of a robust and highly efficient server-side application.

The main feature of the application is a core module that contains the matching algorithm. This algorithm is crucial in calculating the amount of compatibility between users, utilizing the lifestyle choices, profile and personal details. The algorithm's technique serves as the basis for creating roommate matches, guaranteeing that users are paired with peoples who possess comparable interests and beliefs. This strategic approach facilitates the development of harmonious living arrangements, cultivating an atmosphere in which users may participate in shared living experiences that are in line with their preferences and values.

The design stage of Minglemates serves as the foundation around which the entire application is built. This crucial stage coordinates a smooth and integrated series of features that together determine both the user experience and the operational structure of the platform. Each subtle aspect in the design process, starting with the generation of module concepts to the creation of wireframe representations and the careful choice of tools and technologies, played a crucial role in developing the overall structure of Minglemates. Design section provided a detailed examination of the careful process of designing, giving a thorough overview that covers everything from the initial idea to the visualization of user experiences, the creation of the compatibility calculation algorithm, and the complexities of database design.

The beginning of the design phase was a methodical process of generating ideas and conceptualizing the several modules that are crucial for Minglemates. This phase was not only a creative thinking activity, but a deliberate effort to synchronise the platform's features with user expectations. The aim and functioning of each module were carefully examined to ensure that they effectively contribute to the broader goals of fostering meaningful connections and supporting harmonious shared living experiences.

Wireframing has become a crucial element of the design process, providing a visual plan of the application's interface and functionality. The skeleton representations allowed the design team to visually understand the user path, which helped them improve and perfect the layout for the best user engagement. The iterative nature of this stage facilitated ongoing improvement, guaranteeing that the design adheres to user-centric concepts and usability requirements.

Selecting the most appropriate tools and technology was a crucial strategic choice for the development of Minglemates. The alignment of the selected technology with the desired user experience was of utmost importance. The decision to use React.js for the frontend and Java Spring Boot for the backend was not random, but a deliberate option made after careful consideration of the requirement for a technologically advanced stack that is both responsive, dynamic, and resilient. The careful and detailed planning of user experiences was a primary focus in the design process. With the use of the routes users will navigate inside the application helped the development of an instinctive and smooth experience. The user experience was meticulously designed to meet user expectations and achieve the main objective of facilitating meaningful interactions. This involved carefully planning the onboarding process and the complex process of matching roommates.

The development of the compatibility calculation method was a complex endeavor. The method was designed to calculate compatibility levels accurately by utilizing lifestyle preferences and personal information. The algorithmic expertise of Minglemates serves as the foundation of the platform, guaranteeing that roommate matches beyond superficial commonalities and explore intricate lifestyle preferences, fostering the development of enduring friendships. The approach to database architecture prioritized scalability, data integrity, and rapid retrieval. The database's structure and linkages were carefully designed to adapt to the growing requirements of Minglemates as it expands, guaranteeing smooth data administration and retrieval to enhance user experiences.

The careful evaluation procedure carried out on the application has yielded a thorough assessment of its many aspects, guaranteeing that each element functions seamlessly and adheres to the set design criteria. The thorough examination of Application Programming Interfaces (APIs) confirmed the application's proficiency in efficient server communication, user data retrieval, testimonial management, compatibility score calculation, and seamless user registration facilitation. The adherence to preset norms was checked using form validation tests, assuring data consistency and precision in the registration, login, profile setup, preferences setting, and lifestyle information input operations.

The general robustness of all application components was strengthened through functionality testing, with any found deviations swiftly addressed and fixed during the testing phase. The favorable results of these tests indicate that the application, built with React.js and Java Spring Boot, is in line with its design goals and operates as planned. Regarding the assessment of overall performance, the application's dependability and accuracy are confirmed by the successful completion of certain tests.

6.1 ADVANTAGES OF MINGLEMATES

Minglemates, being a Roommate Matching Platform, possesses several benefits that enhance its attractiveness and efficacy in the domain of shared living arrangements. Here are some prominent benefits:

- **Roommate Matching process**: Minglemates excels in its advanced matching process. The application provides very precise roommate matching by taking into account lifestyle choices, personal hobbies, and room preferences. This guarantees that users are linked with persons who possess like beliefs and interests, promoting compatibility in communal living situations.

- **User-Centric Design**: Minglemates places a high priority on user experience by employing a design approach that revolves on the needs and preferences of the users. The thorough planning of user journeys, interface design, and navigation all lead to an application that is not only highly useful but also immensely pleasurable to use. This user-centric methodology improves user engagement and happiness.
- The integration of real-time communication features in the platform enables meaningful conversations among possible housemates. This functionality facilitates effective communication among users, promoting bonds and enabling individuals to familiarize themselves with one another prior to engaging in shared living arrangements.
- **Customization and Preferences**: Minglemates enables users to assert their preferences and enhance roommate matches by tailoring them to their personal tastes. This degree of customization increases the user experience, enabling users to adapt their living arrangements to match their own lifestyle choices and tastes.
- The use of cookies to establish secure sessions was essential for enhancing user privacy and safeguarding data security. Sessions enable Minglemates to securely handle and verify user interactions. Cookies, as a component of session management, aid in monitoring user actions and preferences, facilitating a smooth and customized user experience.
- Minglemates is developed to possess scalability and reliability, utilizing technologies such as React.js and Java Spring Boot. This guarantees that the platform is capable of managing expanding user populations while upholding optimal performance. The selection of these technologies enhances the overall stability of the application.
- The comprehensive testing method used during the development phase guarantees the strength and dependability of Minglemates. The application is subjected to a thorough assessment, including API testing, form validation, and functionality testing, in order to detect and resolve any possible flaws, ensuring a superior user experience.

6.2 SCOPE OF ENHANCEMENTS

- **Algorithm Robustness**: Although the matching algorithm is the central component of Minglemates, there is potential for improvement to boost its resilience. By continuously refining and integrating machine learning components, the accuracy of compatibility could be further enhanced.

- **Preference options**: It is essential to include a broader range of preference choices to improve accuracy in roommate matching. By enabling consumers to offer more elaborate insights into their preferences, a more precise compatibility evaluation may be achieved.
- **Email Verification**: While Minglemates already places a high importance on user security, implementing email verification will enhance the level of authenticity even more. Implementing email ID verification throughout the signup process would reduce the likelihood of fraudulent accounts and improve the platform's credibility.
- Implementing a group chat function might be advantageous in situations where users are unsure about choosing amongst proposed housemates for decision-making purposes. This would allow potential roommates to participate in collaborative talks, promoting a transparent process of decision-making.
- Implementing server-side sessions is crucial for enhancing security measures. By storing session data securely on the server, the risks related to client-side storage are minimized, hence improving the overall security of the system.
- Enhancing data security requires implementing measures such as regulating the usage of GET methods and closely examining the transfer of request header data. Enforcing more stringent protocols for data retrieval and transmission between the server and client improves defense against potential security vulnerabilities.

At its core, Minglemates is a versatile and user-focused platform for matching roommates. It not only meets the present needs of shared living but also has a proactive approach to ongoing enhancement. The comprehensive testing protocols, advanced matching algorithm, and intuitive design highlight its dedication to delivering a dependable and pleasurable experience. Recognizing its limitations, the platform's plan for the future include improving the matching algorithm, broadening preferences choices, and including extra security measures, such as email verification and strong session management. These actions are in line with Minglemates' commitment to being flexible and guaranteeing user satisfaction. These steps align with Minglemates' dedication to adaptability and user satisfaction, ensuring that the platform not only meets user expectations. Minglemates is committed to developing and improving its platform to facilitate genuine connections, improve the accuracy of compatibility matching, and provide a safe and lively community for individuals seeking friendly shared living experiences.

REFERENCES

1 Zana, A. (2020) The importance of roommates, Authentic Travel.And. Available at: https://www.authentictraveland.com/post/the-importance-of-roommates

2 Freeze, P. (2023) Pros and cons of living with a roommate, Bay Property Management Group. Available at: https://www.baymgmtgroup.com/blog/living-with-a-roommate/#when

3 Pekarsky, M. (2020) Does your web app need a front-end framework?, Stack Overflow. Available at: https://stackoverflow.blog/2020/02/03/is-it-time-for-a-front-end-framework/

4 Dhaduk, H. (2024) Best frontend frameworks for web development in 2023, Simform. Available at: https://www.simform.com/blog/best-frontend-frameworks/#:~:text=Frontend%20frameworks%20are%20collections%20of,utilities%20to%20streamline%20the%20process

5 Rawat, P. and Mahajan, A.N., 2020. ReactJS: A modern web development framework. *International Journal of Innovative Science and Research Technology*, *5*(11), pp.698-702. Available at https://ijisrt.com/assets/upload/files/IJISRT20NOV485.pdf

6 Aggarwal, S. (2018) Modern Web-Development using ReactJS. Available at: http://ijrra.net/Vol5issue1/IJRRA-05-01-27.pdf

7 Herbert, D. (2023) What is react.js? uses, examples, & more, HubSpot Blog. Available at: https://blog.hubspot.com/website/react-js

8 Roommates.com, LLC (ed.) (2023) About Us, Roommates.com. Available at: https://www.roommates.com/about-us.

9 SpareRoom (2023) How spareroom works. Available at: https://www.spareroom.co.uk/content/about-us/how-spareroom-works

10 Geetha, G. et al. (2022) 'Interpretation and analysis of Angular Framework', 2022 International Conference on Power, Energy, Control and Transmission Systems (ICPECTS)] Available at : https://ieeexplore.ieee.org/document/10047474

11 Sanity (2023) What is angular in web development? - framework Overview - glossary, Sanity.io. Available at: https://www.sanity.io/glossary/angular

12 Cetin, V. (2023) What are angular's advantages as a front-end framework?, Devoteam. Available at: https://www.devoteam.com/expert-view/angular-front-end-framework/

13 Tran, N. (2020) Applying vue.js framework in developing web applications, Theseus. Available at: https://www.theseus.fi/handle/10024/346386

14 Tuama, D.Ó. (2022) What is vue.js?, Code Institute Global. Available at: https://codeinstitute.net/blog/what-is-vue-js/

15 Vue.js (2023) Vue.js, Introduction | Vue.js. Available at: https://vuejs.org/guide/introduction.html

16 Saks, E., 2019. JavaScript Frameworks: Angular vs React vs Vue. Available at: https://www.theseus.fi/bitstream/handle/10024/261970/Thesis-Elar-Saks.pdf

17 Daityari, S. (2023) Angular vs react vs Vue: Which framework to choose , CodeinWP. Available at: https://www.codeinwp.com/blog/angular-vs-vue-vs-react/#gref

18 Fagbuyiro , D. (2023) Why you should use java for backend development, freeCodeCamp.org. Available at: https://www.freecodecamp.org/news/java-for-backend-web-development/.

19 Suryotrisongko, H., Jayanto, D.P. and Tjahyanto, A. (2017) 'Design and development of backend application for public complaint systems using Microservice Spring Boot', Procedia Computer Science, 124, pp. 736–743. Available at: https://www.sciencedirect.com/science/article/pii/S1877050917329800

20 Sheldon, R., 2023. model-view-controller (MVC). [online] WhatIs. Available at: https://www.techtarget.com/whatis/definition/model-view-controller-MVC

APPENDICES

APPENDIX A : Ethics training certificate

APPENDIX B: Abbreviations

1. DB – Database

APPENDIX C: Project code

Roommate-matching-platform.zip

File:

roommate_db.sql (DB file)

Folders:

front-end (contains front end code)

back-end (contains back-end code)